PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : H04L 25/02 — A1

(11) International Publication Number: WO 98/19431

(43) International Publication Date: 7 May 1998 (07.05.98)

(21) International Application Number: PCT/US97/19676

(22) International Filing Date: 27 October 1997 (27.10.97)

(30) Priority Data:
08/741,273 30 October 1996 (30.10.96) US

(71) Applicant: QUALCOMM INCORPORATED [US/US]; 6455 Lusk Boulevard, San Diego, CA 92121 (US).

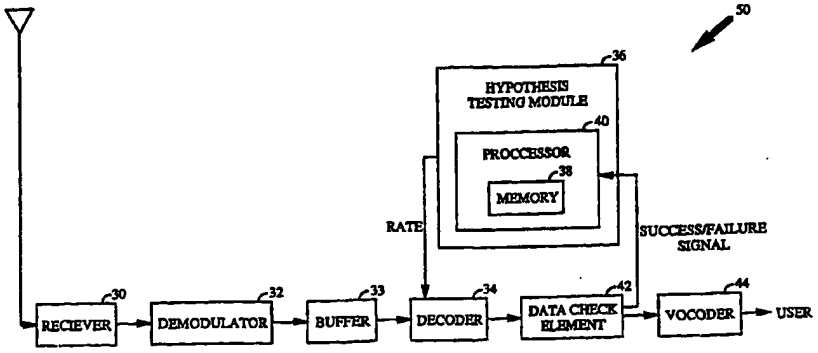
(72) Inventors: TIEDEMANN, Edward, G., Jr.; 4350 Bromfield Avenue, San Diego, CA 92122 (US). LIN, Yu-Chuan; 585 W. 63rd Avenue, Vancouver, British Columbia V6P 2G7 (CA).

(74) Agents: OGROD, Gregory, D. et al.; Qualcomm Incorporated, 6455 Lusk Boulevard, San Diego, CA 92121 (US).

(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, HU, ID, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW, ARIPO patent (GH, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: METHOD AND APPARATUS FOR DECODING VARIABLE RATE DATA



RECIEVING SYSTEM

(57) Abstract

A system and method for determining the data rate of a frame of data at a receiver (50) of a variable rate communications system. A vocoder at a transmitter encodes a frame of data at one of the rates of a predetermined set of rates. The data rate is dependent on the speech activity during the time frame of the data. The data frame is also formatted with overhead bits, including bits for error detection and detection. At the receiver (50), the data rate for the frame is determined based on hypothesis testing. Because the data rate is based on speech activity, a hypothesis test may be designed based on the statistics of speech activity. The received data frame is first decoded by a decoder (34) into information bits at the most probable rate as provided by the hypothesis testing module (36). Data check element (42) generates error metrics for the decoded information bits. If the error metrics indicate that the information bits are of good quality, then the information bits are presented to a vocoder (44) at the receiver to be processed for interface with the user. If the error metrics indicate that the information bits have not been properly decoded, then decoder (34) decodes the received data frame at the other rates of the set of rates until the actual data rate is determined.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

METHOD AND APPARATUS FOR DECODING VARIABLE RATE DATA

BACKGROUND OF THE INVENTION

I. Field of the Invention

The present invention relates to digital communications. More particularly, the present invention relates to a novel and improved system and method for determining, at a receiver of a variable rate communication system, the rate at which data has been encoded for transmission.

II. Description of the Related Art

The use of code division multiple access (CDMA) modulation techniques is one of several techniques for facilitating communications in which a large number of system users are present. Although other techniques such as time division multiple access (TDMA), frequency division multiple access (FDMA), and AM modulation schemes such as amplitude companded single sideband (ACSSB) are known, CDMA has significant advantages over these other techniques. The use of CDMA techniques in a multiple access communication system is disclosed in U.S. Pat. No. 4,901,307, entitled "SPREAD SPECTRUM MULTIPLE ACCESS COMMUNICATION SYSTEM USING SATELLITE OR TERRESTRIAL REPEATERS," assigned to the assignee of the present invention and incorporated by reference herein.

CDMA systems often employ a variable rate vocoder to encode data so that the data rate can be varied from one data frame to another. An exemplary embodiment of a variable rate vocoder is described in U.S. Pat. No. 5,414,796, entitled "VARIABLE RATE VOCODER," assigned to the assignee of the present invention and incorporated by reference herein. The use of a variable rate communications channel reduces mutual interference by eliminating unnecessary transmissions when there is no useful speech to be transmitted. Algorithms are utilized within the vocoder for generating a varying number of information bits in each frame in accordance with variations in speech activity. For example, a vocoder with a set of four rates may produce 20 millisecond data frames containing 16, 40, 80, or 171 information bits, depending on the activity of the speaker. It is desired to transmit each data frame in a fixed amount of time by varying the transmission rate of communications.

Additional details on the formatting of the vocoder data into data frames are described in U.S. Pat. No. 5,511,073, entitled "METHOD AND APPARATUS FOR THE FORMATTING OF DATA FOR TRANSMISSION," assigned to the assignee of the present invention and herein incorporated by reference. The data frames may be further processed, spread spectrum modulated, and transmitted as described in U.S. Pat. No. 5,103,459, entitled "SYSTEM AND METHOD FOR GENERATING WAVEFORMS IN A CDMA CELLULAR TELEPHONE SYSTEM," assigned to the assignee of the present invention and incorporated by reference herein.

Variable rate systems can be developed which include explicit rate information. If the rate is included as part of a variable rate frame, then the rate is not recoverable until after the frame has already been properly decoded, at which point the rate has already been determined. Rather than including the rate in a variable rate frame, the rate could instead be sent in a non-variable rate portion of the frame. However, only a few bits are typically needed to represent the rate, and these bits cannot be efficiently encoded and interleaved in order to provide error protection for fading communications channels. Furthermore, the rate information is only available after some decoding delay and are subject to error.

Alternatively, variable rate systems can be developed which do not include explicit rate information. One technique for the receiver to determine the rate of a received data frame where the rate information is not explicitly included in the frame is described in copending U.S. Patent Application Serial No. 08/233,570, entitled "METHOD AND APPARATUS FOR DETERMINING DATA RATE OF TRANSMITTED VARIABLE RATE DATA IN A COMMUNICATIONS RECEIVER," filed April 26, 1994, assigned to the assignee of the present invention, and incorporated by reference. Another technique is described in copending U.S. Patent Application Serial No. 08/126,477, entitled "MULTIRATE SERIAL VITERBI DECODER FOR CODE DIVISION MULTIPLE ACCESS SYSTEM APPLICATIONS," filed Sept. 24, 1993, assigned to the assignee of the present invention, and incorporated by reference. According to these techniques, each received data frame is decoded at each of the possible rates. Error metrics, describing the quality of the decoded symbols for each frame decoded at each rate, are provided to a processor. The error metrics may include Cyclic Redundancy Check (CRC) results, Yamamoto Quality Metrics, and Symbol Error Rates. These error metrics are well-known in communications systems. The processor analyzes the error metrics and

determines the most probable rate at which the incoming symbols were transmitted.

Decoding each received data frame at each possible data rate will eventually generate the desired decoded data. However, the search through all possible rates is not the most efficient use of processing resources in a receiver. Also, as higher transmission rates are used, power consumption for determining the transmission rate also increases. This is because there are more bits per frame to be processed. Furthermore, as technology evolves, variable rate systems may utilize larger sets of data rates for communicating information. The use of larger sets of rates will make the exhaustive decoding at all possible rates infeasible. In addition, the decoding delay will not be tolerable for some system applications. Consequently, a more efficient rate determination system is needed in a variable rate communications environment. These problems and deficiencies are clearly felt in the art and are solved by the present invention in the manner described below.

SUMMARY OF THE INVENTION

The present invention is a novel and improved system and method for determining the transmission rate of communications in a variable rate communications system. In a variable rate system, the data rate at which a data frame is encoded may be based on the speech activity during the time frame. Because the characteristics of speech are known, probability functions may be defined for the data rates which are dependent on the characteristics of speech. The probability functions may in addition be dependent on the measured statistics of the received data frames. Furthermore, hypothesis tests can be designed based on the probability functions to determine the most likely data rate of a received frame of data. These probability functions may be dependent on the selected service option. For example, the probability functions for data services will be different than for voice services.

At the receiver of the present invention, a processor causes a decoder to decode the received frame of data into information bits at the most probable rate as determined by the hypothesis test. The most probable rate may, for example, be the rate of the previous frame of data. The decoder also generates error metrics for the decoded information bits. The decoded bits and the error metrics are provided to a data check element which checks the decoded bits for correctness. If the error metrics indicate that the

decoded information bits are of good quality, then the information bits are provided to a vocoder which further processes the data and provides speech to the user. Otherwise, a failure signal is presented to the processor. The processor then causes the decoder to decode the received frame of data at other data rates until the correct data rate is found.

BRIEF DESCRIPTION OF THE DRAWINGS

The features, objects, and advantages of the present invention will become more apparent from the detailed description set forth below when taken in conjunction with the drawings in which like reference characters identify correspondingly throughout and wherein:

FIG. 1 is a schematic overview of an exemplary CDMA cellular telephone system;

FIG. 2 is a block diagram of a variable rate receiving system with particular reference to the rate determination features of the present invention;

FIGS. 3 and 4 are flow charts illustrating two embodiments of the processing steps involved in rate determination wherein the hypothesis test designates the rate of the previous frame of data as the most probable rate for the current frame of data;

FIGS. 5 and 6 are flow charts illustrating two embodiments of the processing steps involved in rate determination wherein the hypothesis test is based on the a priori probability distribution of the data rates; and

FIGS. 7 and 8 are flow charts illustrating two embodiments of the processing steps involved in rate determination wherein the hypothesis test is based on the conditional probability distribution of the data rates.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

An exemplary cellular mobile telephone system in which the present invention is embodied is illustrated in FIG. 1. For purposes of example this system is described herein within the context of a CDMA cellular communications system. However, it should be understood that the invention is applicable to other types of communication systems such as personal communication systems (PCS), wireless local loop, private branch exchange (PBX) or other known systems. Furthermore systems utilizing other well known transmission modulation schemes such as TDMA and

FDMA as well as other spread spectrum systems may employ the present invention.

An exemplary cellular system in which the rate determination system of the present invention may be implemented is illustrated in FIG. 1. In FIG. 1, system controller and switch **10** typically include appropriate interface and processing hardware for providing system control information to the cell-sites. Controller **10** controls the routing of telephone calls from the public switched telephone network (PSTN) to the appropriate cell-site for transmission to the appropriate mobile unit. Controller **10** also controls the routing of calls from the mobile units via at least one cell-site to the PSTN. Controller **10** may direct calls between mobile users via the appropriate cell-site stations since such mobile units do not typically communicate directly with one another.

Controller **10** may be coupled to the cell-sites by various means such as dedicated telephone lines, optical fiber links or by radio frequency communications. In FIG. 1, two exemplary cell-sites, **12** and **14**, along with two exemplary mobile units, **16** and **18**, which include cellular telephones, are illustrated. Arrows **20a-20b** and **22a-22b** respectively define the possible communication links between cell-site **12** and mobile units **16** and **18**. Similarly, arrows **24a-24b** and arrows **26a-26b** respectively define the possible communication links between cell-site **14** and mobile units **18** and **16**.

The cellular system illustrated in FIG. 1 may employ a variable rate data channel for communications between cell-sites **12**, **14** and mobile units **16**, **18**. By example, a vocoder (not shown) may encode sampled voice information into symbols at four different rates according to the IS-95-A standard. The IS-95-A Mobile Station-Base Station Compatibility Standard for Dual Mode Wideband Spread Spectrum Cellular System has been provided by the telecommunications industry association (TIA) for CDMA communications. According to IS-95-A, speech is encoded at approximately 8,550 bits per second (bps), 4,000 bps, 2,000 bps, and 800 bps based on voice activity during a 20 millisecond (ms) frame of data. Each frame of vocoder data is then formatted with overhead bits as 9,600 bps, 4,800 bps, 2,400 bps, and 1,200 bps data frames for transmission. The 9,600 bps frame is referred to as a full rate frame; the 4,800 bps data frame is referred to as a half rate frame; a 2,400 bps data frame is referred to as a quarter rate frame; and a 1,200 bps data frame is referred to as an eighth rate frame. Although this example describes a set of four data rates of the IS-95-A standard, it should be recognized that the present invention is equally applicable in systems

utilizing different transmission rates and/or a different number of variable rates.

By encoding each frame of data based on speech activity, data compression is achievable without impacting the quality of the reconstructed speech. Since speech inherently contains periods of silence, i.e. pauses, the amount of data used to represent these periods can be reduced. Variable rate vocoding most effectively exploits this fact by reducing the data rate for these periods of silence. In a system with a set of four rates as described above, periods of active speech will generally be encoded at full rate, while periods of silence will generally be encoded at eighth rate. Most frames (about 80-90%) are encoded at full or eighth rate. Transitions between active speech and periods of silence will typically be encoded at half or quarter rate. An exemplary encoding technique which compresses data based on speech activity is described in U.S. Pat. No. 5,511,073 mentioned above.

The data frames are also formatted with overhead bits, which generally will include additional bits for error correction and detection, such as Cyclic Redundancy Check (CRC) bits. The CRC bits can be used by the decoder to determine whether or not a frame of data has been received correctly. CRC codes are produced by dividing the data block by a predetermined binary polynomial as is described in detail in IS-95-A.

In a preferred embodiment, each frame of symbol data is interleaved by an interleaver, preferably on a bit level basis, to increase time diversity for purposes of error detection. The formatted data frames undergo further processing, which include modulation, frequency upconversion to the radio frequency (RF) and amplification of the signals of data frames, before transmission.

When signals of the variable rate data frames are received by a receiver, the receiver must determine the rate of transmission in order to properly decode the signals. However, the rate of the received frame is not known by the mobile station a priori. Therefore, some other method of ascertaining the rate is necessary.

The present invention accomplishes rate determination through the use of hypothesis testing. Hypothesis tests are designed based on the probability distribution of the data rates of the frames of speech. Although the data rate of each received frame is not known a priori, the probability of receiving a frame at a given rate can be determined. As mentioned above, a variable rate vocoder encodes each frame of speech at one of a set of predetermined rates based on the speech activity during the time frame.

Since the characteristics of speech activity can be modeled, probabilistic functions of the data rates which depend on speech activity can be derived from the model. Hypothesis tests can then be designed based on the probabilistic functions of data rates to determine the most likely data rate for each received frame of data.

The use of hypothesis testing for rate determination in a variable rate receiving system may be better appreciated by referring to FIG. 2. In a CDMA environment, for example, the receiving system 50 of FIG. 2 may be implemented in either a mobile unit or a cell site in order to determine the data rate of received signals. The present invention offers particular advantages because it avoids the exhaustive decoding at all rates. By choosing a hypothesis and checking the hypothesis for correctness, the average amount of processing for each received frame is reduced. This is especially important in the mobile unit because reduced processing, and thereby power consumption, in the decoding process can extend battery life in the receiver.

The variable rate receiving system 50 illustrated in FIG. 2 includes receiver 30 for collecting transmitted signals, including the data signal of interest. Receiver 30 amplifies and frequency downconverts the received signals from the RF frequency band to the intermediate frequency (IF) band.

The IF signals are presented to demodulator 32. The design and implementation of demodulator 32 are described in detail in U.S. Pat. No. 5,490,165, entitled "DEMODULATION ELEMENT ASSIGNMENT IN A SYSTEM CAPABLE OF RECEIVING MULTIPLE SIGNALS," issued Feb. 6, 1996, and assigned to the assignee of the present invention, the disclosure of which is incorporated by reference herein. Demodulator 32 demodulates the IF signal to produce a data signal consisting of the symbols of one frame of data. Demodulator 32 generates the data signal by despreading and correlating the IF signal addressed to the receiver. The demodulated data signal is then fed to buffer 33. Buffer 33 stores the demodulated data signal, or the received symbols, until it is properly decoded. Buffer 33 may also be the deinterleaver if the data frame had been interleaved for transmission. Buffer 33 provides the demodulated symbol data to decoder 34.

Hypothesis testing module 36 implements the hypothesis test for determining the data rate of a received frame of data. Hypothesis testing module 36 comprises processor 40, which includes memory 38. The information needed in hypothesis testing such as the decoded rates from the previous frames and the probabilities are stored in memory 38. For each data frame received, processor 40 determines the most probable rate based

on the information stored in memory 38. Processor 40 then presents the most probable data rate to decoder 34 which decodes the data signal at this most probable rate to produce decoded bits.

In the exemplary embodiment, decoder 34 is a trellis decoder capable of decoding data of varying rates, such as a Viterbi decoder. The design and implementation of a multirate Viterbi decoder which exhaustively decodes a received signal at all rates of a set of rates is described in the aforementioned U.S. Patent Applications 08/233,570 and 08/126,477. It will be understood by one skilled in the art that the multirate Viterbi decoder may be modified to decode at a selected rate. This may be accomplished by having the Viterbi decoder receive a rate indicator input, in response to which the decoder decodes the data signal according to the rate indicator. Thus, the modified Viterbi decoder may decode a received data frame based on a rate indicator supplied by processor 40 of hypothesis testing module 36.

Decoder 34 generates information data bits and error metrics characterizing the information bits. The error metrics include the previously described CRC bits, which were added into the data frames as overhead bits. Decoder 34 may also generate other error metrics, such as the Yamamoto Quality Metric and the Symbol Error Rate (SER). The Yamamoto metric is determined by comparing the differences in the metrics of remerging paths in each step of the Viterbi decoding with a threshold and labeling a path as unreliable if the metric difference is less than a quality threshold. If the final path selected by the Viterbi decoder has been labeled as unreliable at any step, the decoder output is labeled as unreliable. Otherwise, it is labeled as reliable. The Symbol Error Rate is determined by taking the decoded bits, re-encoding these bits to provided re-encoded symbols, and comparing these re-encoded symbols against the received symbols which are stored in buffer 33. The SER is a measure of the mismatching between the re-encoded symbols and the received symbols. The decoded information bits and the error metrics are provided to data check element 42, which determines if the information bits have been correctly decoded.

In a preferred embodiment, data check element 42 first checks the CRC bits. If the CRC check fails, then data check element 42 provides a signal indicative of the failure to processor 40. If the CRC check passes, then data check element 42 determines if the re-encoded SER is below a certain threshold. If the SER is above the threshold, then a signal indicative of failure is provided to processor 40. Otherwise, the data rate provided by hypothesis testing module 36 is determined to be correct, and a success

signal is provided to processor 40, whereupon no further decoding is performed on the data frame. The properly decoded data signal is presented to variable rate vocoder 44.

When processor 40 receives a failure signal indicating that data symbols have not been properly decoded into information bits, processor 40 will determine at least one other data rate from the set of data rates at which to decode the data symbols. Processor 40 provides the rate information to decoder 34, which decodes the data symbols at the rate provided. For each data rate at which the data signal is decoded, data check element 42 will determine the quality of the decoded information bits. Upon determination by data check element 42 that the correct data rate has been found, a signal of decoded information bits is provided to variable rate vocoder 44. Vocoder 44 will then process the information bits for interface with the user.

Hypothesis testing module 36 may implement any of a number of hypothesis tests for determining the data rate of a received frame of data. For example the hypothesis test may be based on known statistics of speech activity. It is known that for a set of four rates using 20 ms frames, a full rate frame will usually be followed by another full rate frame, while an eighth-rate frame will usually be followed by another eighth rate frame. Further, it is also known that most frames will either be full or eighth rate rather than half or quarter rate, because the periods of speech and silence do not occur in 20 ms bursts. Based on these characteristics, the hypothesis test may designate the rate of the previous frame of data as the most probable rate for the currently received frame of data.

In an exemplary implementation, the rate of the previous frame of data is stored in memory 38 of hypothesis testing module 36. When a data frame is received, processor 40 of hypothesis testing module 36 obtains the rate of the previous frame from memory 38 and presents it to decoder 34. Decoder 34 decodes the received data frame at the rate of the previous frame to produce information bits. Decoder 34 also generates error metrics which are then presented to data check element 42 along with the information bits. If data check element 42 determines from the error metrics that the decoded bits are of good quality, then the information bits are presented to vocoder 44. Otherwise, a failure indication is sent from data check element 42 to processor 40. Processor 40 may then have decoder 34 exhaustively decode the data frame at all other rates before determining the data rate. A flow chart illustrating some of the steps involved in rate determination as described in the embodiment above is shown in FIG. 3.

Alternatively, processor 40 may have decoder 34 sequentially decode the data frame according to a ranking from the next most likely rate to the least likely rate. The ranking may be determined in a number of ways, such as according to the probability distributions described below. For each decoding, error metrics are generated by decoder 34 and checked by data check element 42 for correctness. When correctly decoded, the decoded frame is passed on to vocoder 44. A flow chart illustrating some of the processing steps of this embodiment is shown in FIG. 4.

Another implementation of hypothesis testing module 36 is based upon the a priori probability distribution of data rates. For a set of four rates, the a priori probability distribution (P) of the data rates may be defined as:

$$P = \text{Prob}\{R_t\}, \tag{3}$$

where R_t refers to the full, half, quarter, or eighth rate at time t. The likelihood of receiving a frame at each of the different data rates of a set of rates are maintained in memory 38 of processor 40. Generally, the probability distribution of the data rates are determined based on the theoretical statistics or the empirical statistics of speech activity. The likelihood of receiving a frame at the different rates are then permanently stored in memory 38 for determining the rate of every received frame of data. In a more sophisticated embodiment, the likelihood of the rates stored in memory 38 may be updated based on the actual statistics of the received frames of data.

For each new frame of data received, processor 40 obtains the most probable rate from memory 38 and presents the most probable rate to decoder 34. Decoder 34 decodes the data signal at this most probable data rate and presents the decoded data to data check element 42. Error metrics, including the CRC, are also generated by decoder 34 and presented to data check element 42. Other error metrics may also be generated for checking by data check element 42. If the error metrics indicate that the decoded bits are of good quality, then the information bits are presented to vocoder 44. Otherwise, a failure indication is sent from data check element 42 to processor 40. Then, processor 40 obtains the second most likely data rate from memory 38 and presents it to decoder 34, and the process of decoding and error checking is continued until the correct data rate is found. A flow chart of the processing steps of this embodiment is illustrated in FIG. 5. Alternatively, upon receipt of a failure signal by processor 40, processor 40 may cause decoder 34 to exhaustively decode the data frame at each of the

other data rates of the set of rates, and error metrics are checked for each decoding in order to determine the actual rate of transmission. A flow chart of the processing steps of this embodiment is illustrated in FIG. 6.

Instead of designing the hypothesis test based on the simple probability distribution of the data rates, conditional probabilities may be used to improve on the accuracy of the rate determination. For example, the probability of receiving a data frame at a given rate may be defined to be conditioned on the actual rates of the previous frames of data. Conditional probabilities based on the previous rates work well because transition characteristics of the data signals are well known. For example, if the rate two frames ago was eighth rate and the rate for the previous frame was half rate, then the most likely rate for the current frame is full rate, because the transition to half rate indicates the onset of active speech. Conversely, if the rate two frames ago was full rate and the rate for the previous frame was quarter rate, then the most likely rate for the present frame might be eighth rate, because the rate transition indicates the onset of silence.

The probability distribution of the data rates conditioned on the rates of the previous n frames of data may be defined as:

$$P = \text{Prob}\{ R_t \mid R_{t-1}, R_{t-2}, \ldots, R_{t-n}\} \quad (4)$$

where R_t again refers to the rate at time t, and R_{t-1}, R_{t-2}, . . ., R_{t-n} refers to rate(s) of the previous n frame(s) of data, for $n \geq 1$. The likelihood of receiving a frame at each of the different data rates of a set of rates conditioned on the previous n actual rates are stored in memory **38** of processor **40**. In addition, the actual data rates of the previous n frames of data are maintained by processor **40**, and may be stored in memory **38** as the rates are determined.

For each received frame of data, processor **40** will determine the most probable data rate conditioned on the previous n actual data rates and present it to decoder **34**. Decoder **34** will decode the frame at this most probable data rate and present the decoded bits to data check element **42**. In addition, error metrics are generated by decoder **34** and presented to data check element **42**. If the error metrics indicate that the decoded bits are of good quality, then the information bits are presented to vocoder **44**. Also, processor **40** is informed of the rate decision so that it can maintain the history of chosen rates. That is, processor **40** is supplied R_t so that it can be used in determining Prob{ $R_t \mid R_{t-1}, R_{t-2}, \ldots, R_{t-n}$} for the next frame. If error metrics indicate an unsuccessful decoding, then a failure indication signal is

sent from data check element **42** to processor **40**, and processor **40** determines the second most probable data rate conditioned on the previous n actual data rates to decode the data frame. As in the simple probabilities case, the process of decoding and error checking is continued until the correct data rate is found. Some of the processing steps of this embodiment are illustrated in a flow chart in FIG. 7. Also as in the simple probabilities case, after a failed decoding at the most likely rate, decoder **34** may exhaustively decode the data frame at all of the other data rates and have error metrics checked for all decoding in order to determine the data rate. Some of the processing steps of this embodiment are illustrated in a flow chart in FIG. 8.

It should be understood that the conditional probability distribution of the data rates may depend on statistics other than the actual rates of the previous frames of data. For example, the probability distribution may be conditioned on one or more frame quality measurements. The probability distribution is then defined to be:

$$P = \text{Prob}\{R_t \mid X_1, X_2, \ldots, X_k\}, \tag{5}$$

where R_t is the rate at time t, and $X_1, X_2, \ldots X_k$ are one or more frame quality measurements. The k frame quality measurements may be measurements performed on the current frame of data, or measurements performed on previous frame(s) of data, or a combination of both. An example of a frame quality measurement is the SER error metric mentioned above. Thus, the probability of receiving a frame at a given rate is conditioned on the SER obtained from the previous decoding if a previous decoding had been performed.

The conditional probability distribution may also depend on a combination of the actual rates of the previous frames of data and the frame quality measurements. In this case, the probability distribution of the data rates is defined as:

$$Pt = \text{Prob}\{ R_t \mid R_{t-1}, R_{t-2}, \ldots, R_{t-n}, X_1, X_2, \ldots, X_k \}, \tag{6}$$

where R_t is the rate at time t, $R_{t-1}, R_{t-2}, \ldots, R_{t-n}$ are the rates of the previous frames of data and $X_1, X_2, \ldots, X_k$ are the frame quality measurements.

In the cases where the probability distribution is based on frame quality measurements, the frame quality measurements should be maintained in processor **40** of hypothesis testing module **36**. As can be seen from the above description, the hypothesized frame rate may be conditioned

on a number of different statistics, and the rates of the previous frames and the frame quality measurements are examples of these statistics. For each data frame received, processor **40** uses the statistics to determine the rate at which to decode the frame.

A further refinement to the determination of the rate at which to decode a received frame of data considers the processing costs of decoding the frame at the various rates in conjunction with hypothesis testing. In this embodiment, an optimum test sequence of the rates is established based on both the probability distribution of the data rates and the cost of decoding at each of the data rates. The optimum test sequence is maintained by processor **40**, which causes decoder **34** to sequentially decode a received frame of data according to the optimum sequence until the correct rate is found. The optimum test sequence is established to minimize the total expected cost of the rate search. Denoting P_i to be the probability that the rate search will stop at test T_i, and C_i to be the cost for conducting test T_i, the total expected cost of the rate search using test sequence $T_1, T_2, \ldots, T_M$, where M is the number of possible rates in the system and $1 \leq i \leq M$, can be modeled as:

$$C_{total} = C_1 * P_1 + (C_1 + C_2)* P_2 + \ldots (C_1 + C_2 + \ldots + C_M)* P_M. \quad (7)$$

The optimum test sequence is found by minimizing the total expected cost C_{total}.

In Equation (7), the cost C_i for conducting test T_i will generally be the processing power required for decoding a frame at the rate specified by test T_i. The cost may be assigned to be proportional to the frame rate specified by the test T_i because the computational complexity of decoder **34** is in general approximately proportional to the number of bits per frame. The probabilities P_i may be assigned by the unconditioned a priori probability distribution of data rates as defined by Equation (3), or any of the conditional probability distributions defined by Equations (4), (5), or (6) above.

In a variable rate communications system where data frames are transmitted at 9,600 bps, 4,800 bps, 2,400 bps, and 1,200 bps, the following example illustrates the formulation of the optimum test sequence for rate determination of a received frame. The costs to decode the 9,600 bps, 4,800 bps, 2,400 bps, and 1,200 bps frames are assumed to be 9.6, 4.8, 2.4, and 1.2, respectively. Further, the probability of receiving a frame at each of the four

rates is assumed to be the unconditioned a priori probabilities having the following values:

$$\text{Prob}(9{,}600 \text{ bps}) = 0.291, \tag{8}$$

$$\text{Prob}(4{,}800 \text{ bps}) = 0.039, \tag{9}$$

$$\text{Prob}(2{,}400 \text{ bps}) = 0.072, \text{ and} \tag{10}$$

$$\text{Prob}(1{,}200 \text{ bps}) = 0.598. \tag{11}$$

The probabilities given in Equations (8)-(11) are derived from steady state empirical data.

A listing of all possible test sequences for rate determination in the system transmitting frames at 9,600, 4,800, 2,400, and 1,200 bps is shown in Table I below. In Table I, column 1 lists all possible test sequences T_1, T_2, T_3, T_4, where $T_i = 1$ refers to the test of decoding at 9,600 bps, $T_i = 1/2$ refers to the test of decoding at 4,800 bps, $T_i = 1/4$ refers to the test of decoding at 2,400 bps, and $T_i = 1/8$ refers to the test of decoding at 1,200 bps. Columns 2 and 3 list the probability P_1 and the cost C_1 of performing the test T_1, columns 4 and 5 list the probability P_2 and the cost C_2 of performing the test T_2, columns 6 and 7 list the probability P_3 and the cost C_3 of performing the test T_3, and columns 8 and 9 list the probability P_4 and the cost C_4 of performing the test T_4. The total cost C_{total} of performing the test sequence T_1, T_2, T_3, T_4 is listed in column 10.

T_1, T_2, T_3, T_4	P_1	C_1	P_2	C_2	P_3	C_3	P_4	C_4	C_{total}
1, 1/2, 1/4, 1/8	0.291	9.6	0.039	4.8	0.072	2.4	0.598	1.2	15.33
1, 1/2, 1/8, 1/4	0.291	9.6	0.039	4.8	0.598	1.2	0.072	2.4	13.98
1, 1/4, 1/2, 1/8	0.291	9.6	0.072	2.4	0.039	4.8	0.598	1.2	15.08
1, 1/4, 1/8, 1/2	0.291	9.6	0.072	2.4	0.598	1.2	0.039	4.8	12.25
1, 1/8, 1/2, 1/4	0.291	9.6	0.598	1.2	0.039	4.8	0.072	2.4	11.16
1, 1/8, 1/4, 1/2	0.291	9.6	0.598	1.2	0.072	2.4	0.039	4.8	10.90
1/2, 1, 1/4, 1/8	0.039	4.8	0.291	9.6	0.072	2.4	0.598	1.2	16.35
1/2, 1, 1/8, 1/4	0.039	4.8	0.291	9.6	0.598	1.2	0.072	2.4	15.00
1/2, 1/4, 1, 1/8	0.039	4.8	0.072	2.4	0.291	9.6	0.598	1.2	16.36
1/2, 1/4, 1/8, 1	0.039	4.8	0.072	2.4	0.598	1.2	0.291	9.6	10.97
1/2, 1/8, 1, 1/4	0.039	4.8	0.598	1.2	0.291	9.6	0.072	2.4	9.61
1/2, 1/8, 1/4, 1	0.039	4.8	0.598	1.2	0.072	2.4	0.291	9.6	9.62
1/4, 1, 1/2, 1/8	0.072	2.4	0.291	9.6	0.039	4.8	0.598	1.2	15.08
1/4, 1, 1/8, 1/2	0.072	2.4	0.291	9.6	0.598	1.2	0.039	4.8	12.26
1/4, 1/2, 1, 1/8	0.072	2.4	0.039	4.8	0.291	9.6	0.598	1.2	16.11
1/4, 1/2, 1/8, 1	0.072	2.4	0.039	4.8	0.598	1.2	0.291	9.6	10.71
1/4, 1/8, 1/2, 1	0.072	2.4	0.598	1.2	0.039	4.8	0.291	9.6	7.89
1/4, 1/8, 1, 1/2	0.072	2.4	0.598	1.2	0.291	9.6	0.039	4.8	6.87
1/8, 1, 1/4, 1/2	0.598	1.2	0.291	9.6	0.072	2.4	0.039	4.8	5.51
1/8, 1, 1/2, 1/4	0.598	1.2	0.291	9.6	0.039	4.8	0.072	2.4	5.76
1/8, 1/2, 1, 1/4	0.598	1.2	0.039	4.8	0.291	9.6	0.072	2.4	6.79
1/8, 1/2, 1/4, 1	0.598	1.2	0.039	4.8	0.072	2.4	0.291	9.6	6.79
1/8, 1/4, 1/2, 1	0.598	1.2	0.072	2.4	0.039	4.8	0.291	9.6	6.54
1/8, 1/4, 1, 1/2	0.598	1.2	0.072	2.4	0.291	9.6	0.039	4.8	5.52

Table I

As shown in Table I, the optimum test sequence is the sequence 1/8, 1, 1/4, 1/2 shown in the 19th row. This test sequence offers the lowest total expected cost of processing. Therefore, the rate determination system would decode a received frame of data at 1,200 bps first. If the decoding at 1.200 bps is not successful, then the frame would be decoded sequentially at 9,600 bps, 2,400 bps, and 4,800 bps until the correct rate is found. In a preferred embodiment, the optimum test sequence is maintained by processor **40** of hypothesis testing module **36**. For each frame of data received, processor **40** causes decoder **34** to decode the frame sequentially according to the optimum test sequence, with each decoding checked by data check element **42**, until the correct data rate is found. Processing resources are efficiently utilized in this rate determination system because the decoding is performed sequentially according to an optimum search sequence.

Based on the embodiments described above, it will be understood by one skilled in the art that the present invention is applicable to all systems

in which data has been encoded according to a variable rate scheme and the data must be decoded in order to determine the rate. Even more generally, the invention is applicable to all systems in which the encoded data E is a function of the data D and some key k, and there exists some information in D or E which permits the verification of the correct D by the receiver. The sequence k may be time varying. The encoded data is represented as:

$$E = f(D,k), \tag{1}$$

where k is from a small set K of keys and where some probability function exists on the set of keys. The inverse of the encoding, or the decoding, can be represented as:

$$D = f^{-1}(E,k), \tag{2}$$

where k is chosen so that D is correct.

As an example, assume that D is data composed of fixed-length sequence D1 and fixed length sequence D2 so that D = D1, D2. Sequence D2 is the Cyclic Redundancy Code (CRC) of D1, so that D2 = fcrc(D1). Assume also that the encoding function, f(D,k), is an exclusive-OR of a fixed-length D with the fixed length sequence k. Then, the decoding, $f^{-1}(E,k)$, would be the exclusive-OR of E with the correct k. The correct k is verified by checking whether D2 = fcrc(D1). The correct k can be found by decoding all possible k's in K and then determining whether the CRC check passes. Alternatively, it can be done by sequentially decoding using one k at a time, with no further decoding once the "correct" k is found. According to the present invention, the order of sequential decoding is to be determined by hypothesis testing. A number of hypothesis tests, including the tests described above, may be utilized. The order of sequential decoding may in addition depend on the cost of processing, as described above. The use of hypothesis testing and/or cost functions in formulating a test sequence for rate determination reduces the average amount of processing as fewer k's will have to be tried.

The previous description of the preferred embodiments is provided to enable any person skilled in the art to make or use the present invention. The various modifications to these embodiments will be readily apparent to those skilled in the art, and the generic principles defined herein may be applied to other embodiments without the use of the inventive faculty. Thus, the present invention is not intended to be limited to the

embodiments shown herein but is to be accorded the widest scope consistent with the principles and novel features disclosed herein.

CLAIMS

1. In a variable rate communications system, a sub-system for determining, at a receiver, the data rate of a received data frame, comprising:

a processor for generating a signal indicating the most likely rate of said received data frame in accordance with a predetermined hypothesis test; and

a decoder for receiving said most likely rate signal and for decoding said received data frame into a decoded frame of bits at said most likely rate.

2. The rate determination sub-system of claim 1 wherein said most likely rate is the rate of the previous data frame.

3. The rate determination sub-system of claim 1 wherein said hypothesis test is based on an a priori probability distribution of data rates.

4. The rate determination sub-system of claim 1 wherein said hypothesis test is based on a conditional probability distribution of data rates conditioned on the rate of at least one previous data frame.

5. The rate determination sub-system of claim 1 wherein said hypothesis test is based on a conditional probability distribution of data rates conditioned on at least one frame quality measurement.

6. The rate determination sub-system of claim 1 further comprising a data check element for receiving said decoded bits, generating error metrics characterizing said decoded bits, and generating a quality indication based on said error metrics for said decoded bits.

7. The rate determination sub-system of claim 6,

further comprising a vocoder for receiving said decoded bits and processing said decoded bits to provide speech to an user upon generation of a positive indication of said quality; and

wherein upon generation of a negative indication of said quality, said processor further causes said decoder to perform additional decoding of said received data frame in accordance with at least one rate other than said most likely rate.

8. The rate determination sub-system of claim 7,

wherein said additional decoding is performed sequentially in accordance with a predetermined test sequence of data rates;

wherein said data check element generates error metrics for each said additional decoding and generates a quality indication based on said error metrics for each said additional decoding; and

wherein said additional decoding terminates upon generation of a positive indication of said quality.

9. The rate determination sub-system of claim 7,

wherein said additional decoding comprises exhaustive decoding of said received data frame at all rates of a rate set except said most likely rate; and

wherein said data check element generates error metrics for each said additional decoding and determines the rate of said received data frame in accordance with said error metrics.

10. The rate determination sub-system of claim 6 wherein said error metrics include a Cyclic Redundancy Check result.

11. The rate determination sub-system of claim 6 wherein said error metrics include a Symbol Error Rate metric.

12. The rate determination sub-system of claim 6 wherein said error metrics include a Yamamoto quality metric.

13. The rate determination sub-system of claim 1 wherein said processor comprises a memory for storing said most likely rate.

14. The rate determination sub-system of claim 1 wherein said decoder is a Viterbi decoder.

15. In a variable rate communications system, a sub-system for determining, at a receiver, the data rate of a received data frame, comprising:

a processor for generating a test sequence of data rates for determining the rate of a received data frame, said test sequence being generated in accordance with a predetermined hypothesis test;

a decoder for decoding said received data frame sequentially according to said test sequence and generating a decoded frame of bits for each rate at which said received data frame is decoded;

a data check element for generating error metrics characterizing said decoded bits and for generating a quality indication based on said error metrics for each rate at which said received data frame is decoded; and

wherein no further decoding is performed upon generation of a positive indication of said quality.

16. The rate determination sub-system of claim 15 wherein said hypothesis test is based on an a priori probability distribution of data rates.

17. The rate determination sub-system of claim 15 wherein said hypothesis test is based on a conditional probability distribution of data rates conditioned on the rate of at least one previous data frame.

18. The rate determination sub-system of claim 15 wherein said hypothesis test is based on a conditional probability distribution of data rates conditioned on at least one frame quality measurement.

19. The rate determination sub-system of claim 16 wherein said test sequence is generated further in accordance with the cost of decoding said received data frame at each of said data rates.

20. The rate determination sub-system of claim 17 wherein said test sequence is generated further in accordance with the cost of decoding said received data frame at each of said data rates.

21. The rate determination sub-system of claim 18 wherein said test sequence is generated further in accordance with the cost of decoding said received data frame at each of said data rates.

22. The rate determination sub-system of claim 15 further comprising a vocoder for receiving said decoded bits and processing said decoded bits to provide speech to an user upon generation of a positive indication of said quality.

23. The rate determination sub-system of claim 15 wherein said error metrics include a Cyclic Redundancy Check result.

24. The rate determination sub-system of claim 15 wherein said error metrics include a Symbol Error Rate metric.

25. The rate determination sub-system of claim 15 wherein said error metrics include a Yamamoto quality metric.

26. The rate determination sub-system of claim 15 wherein said processor comprises a memory for storing said test sequence of data rates.

27. The rate determination sub-system of claim 15 wherein said decoder is a Viterbi decoder.

28. A method for determining the rate of a received data frame in a variable rate communications system, comprising the steps of:

receiving a wide-band signal;

demodulating said wide-band signal to produce a data signal, wherein said data signal has been transmitted at one of a set of possible transmission rates;

generating a test sequence of data rates for determining the rate of said data signal, said test sequence being generated in accordance with a predetermined hypothesis test;

decoding said data signal sequentially according to said test sequence to generate a decoded frame of bits for each rate at which said data signal is decoded;

generating error metrics characterizing said decoded frame of bits for each rate at which said data signal is decoded;

generating a quality indication based on said error metrics for each rate at which said data signal is decoded; and

upon generation of a positive indication of said quality, providing said decoded frame of bits to a vocoder which processes said decoded bits to provide speech to an user.

29. The method of claim 28 wherein said hypothesis test is based on an a priori probability distribution of data rates.

30. The method of claim 28 wherein said hypothesis test is based on a conditional probability distribution of data rates conditioned on the rate of at least one previous data frame.

31. The method of claim 28 wherein said hypothesis test is based on a conditional probability distribution of data rates conditioned on at least one frame quality measurement.

32. The method of claim 29 wherein said test sequence is generated further in accordance with the cost of decoding said received data frame at each of said data rates.

33. The method of claim 30 wherein said test sequence is generated further in accordance with the cost of decoding said received data frame at each of said data rates.

34. The method of claim 31 wherein said test sequence is generated further in accordance with the cost of decoding said received data frame at each of said data rates.

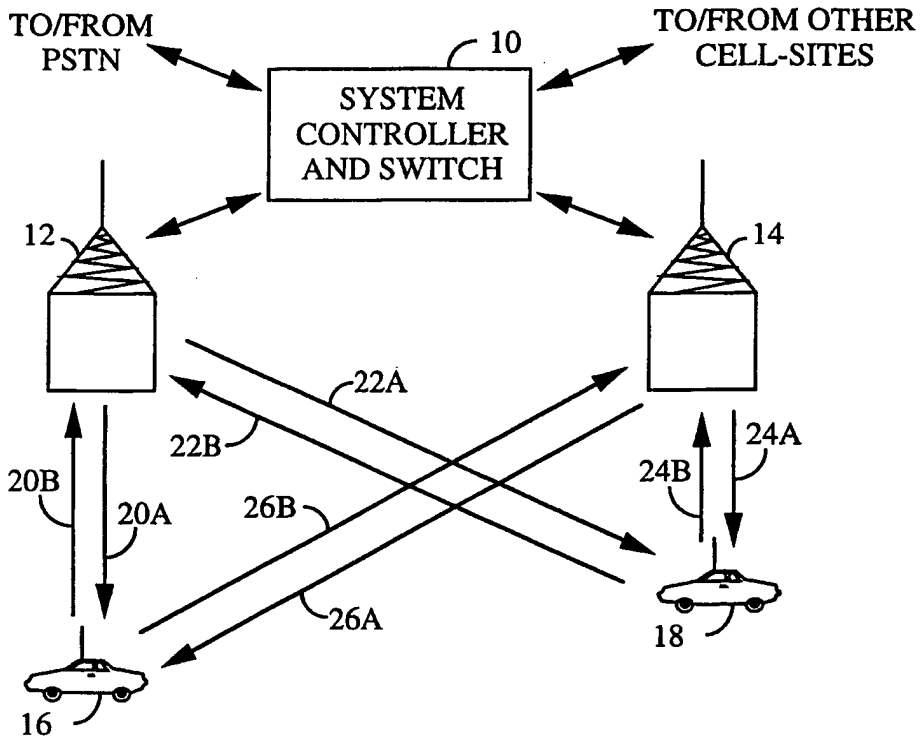

TO/FROM
PSTN
10
TO/FROM OTHER
CELL-SITES
SYSTEM
CONTROLLER
AND SWITCH
12
14
22A
22B
24A
24B
20B
20A
26B
26A
18
16

FIG. 1

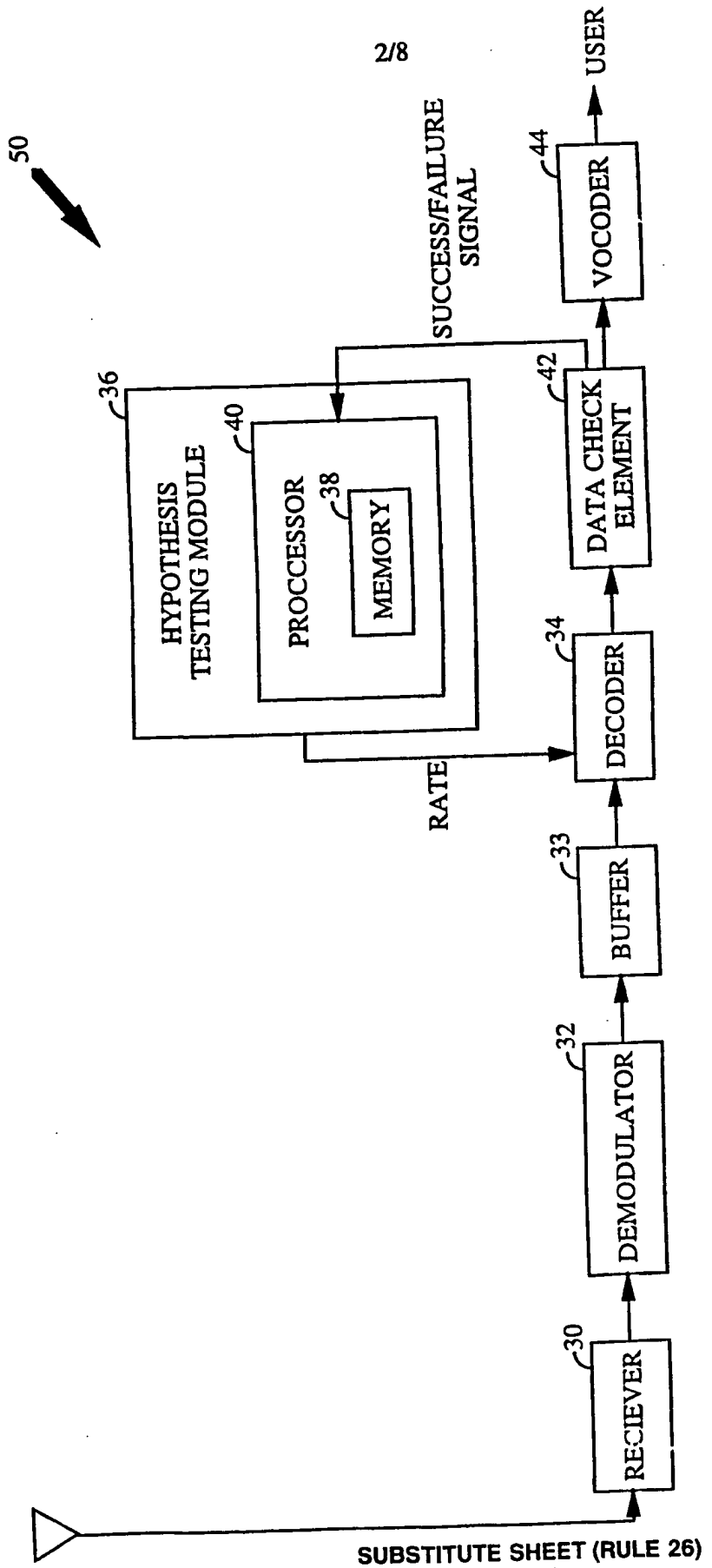

50
SUCCESS/FAILURE SIGNAL
USER
44
VOCODER
42
DATA CHECK ELEMENT
36
HYPOTHESIS TESTING MODULE
40
PROCCESSOR
38
MEMORY
34
DECODER
RATE
33
BUFFER
32
DEMODULATOR
30
RECIEVER
RECIEVING SYSTEM

FIG. 2

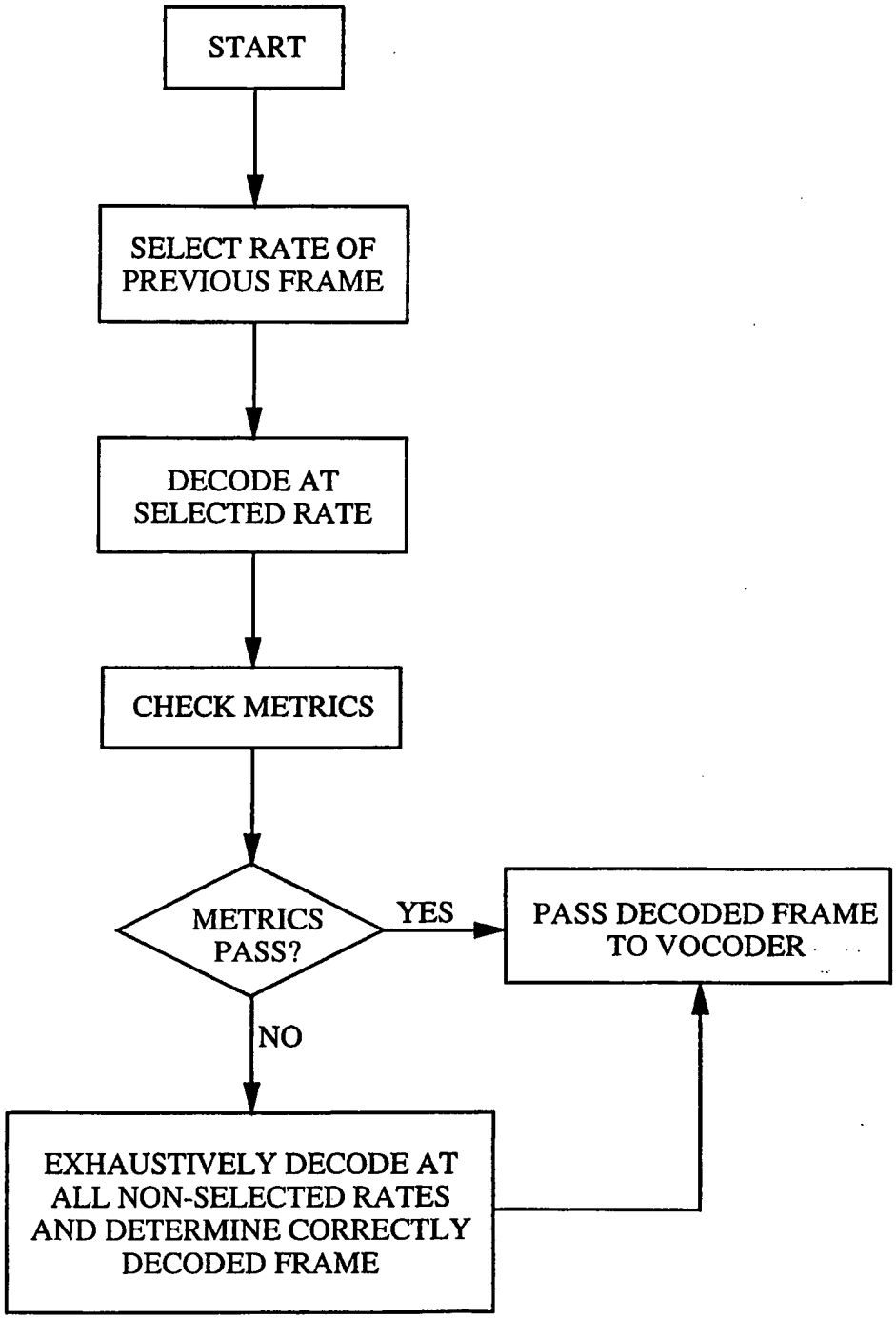

START
SELECT RATE OF PREVIOUS FRAME
DECODE AT SELECTED RATE
CHECK METRICS
METRICS PASS?
YES
PASS DECODED FRAME TO VOCODER
NO
EXHAUSTIVELY DECODE AT ALL NON-SELECTED RATES AND DETERMINE CORRECTLY DECODED FRAME

FIG. 3

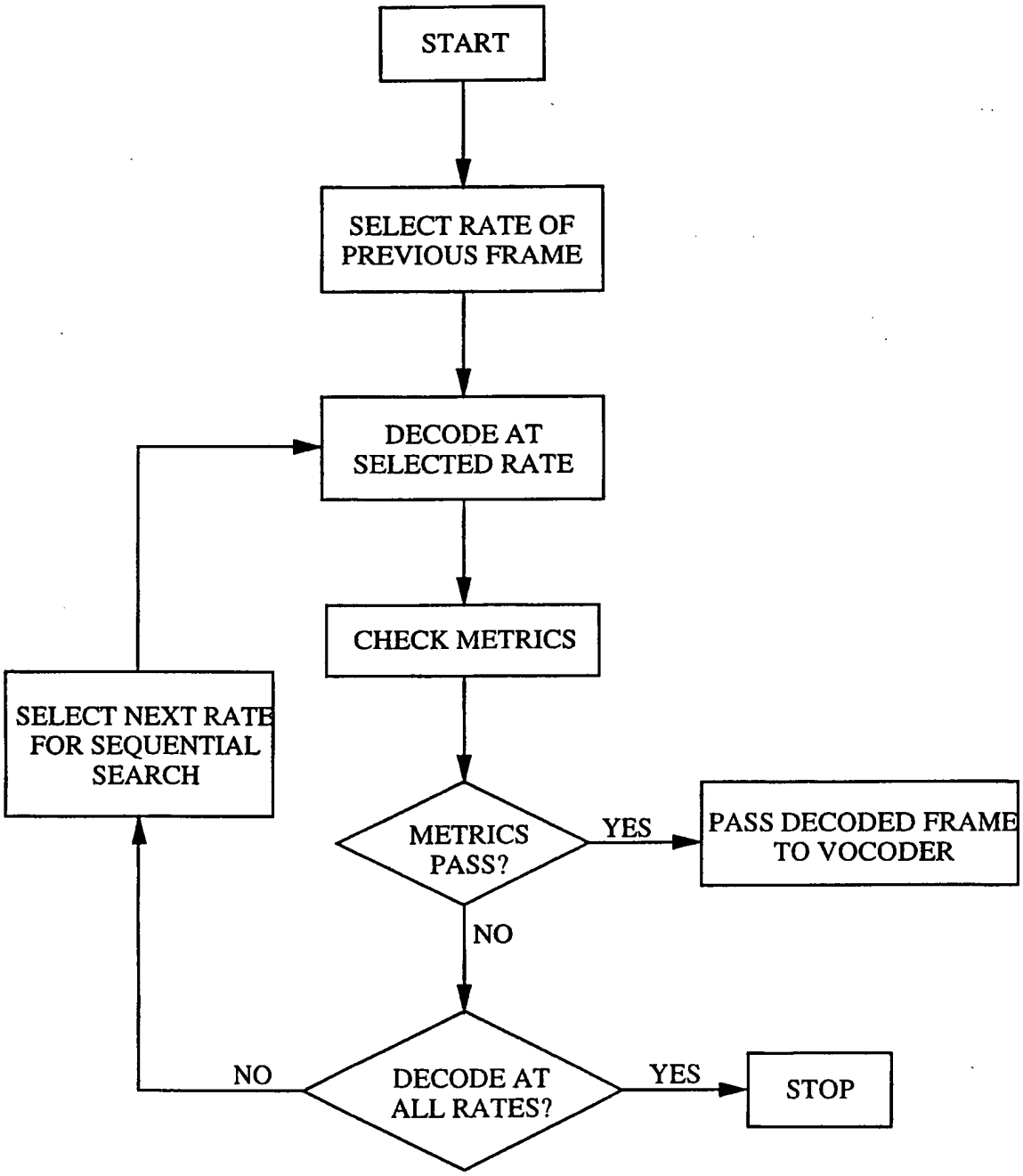

START
SELECT RATE OF PREVIOUS FRAME
DECODE AT SELECTED RATE
CHECK METRICS
METRICS PASS?
YES
PASS DECODED FRAME TO VOCODER
NO
DECODE AT ALL RATES?
NO
SELECT NEXT RATE FOR SEQUENTIAL SEARCH
YES
STOP

FIG. 4

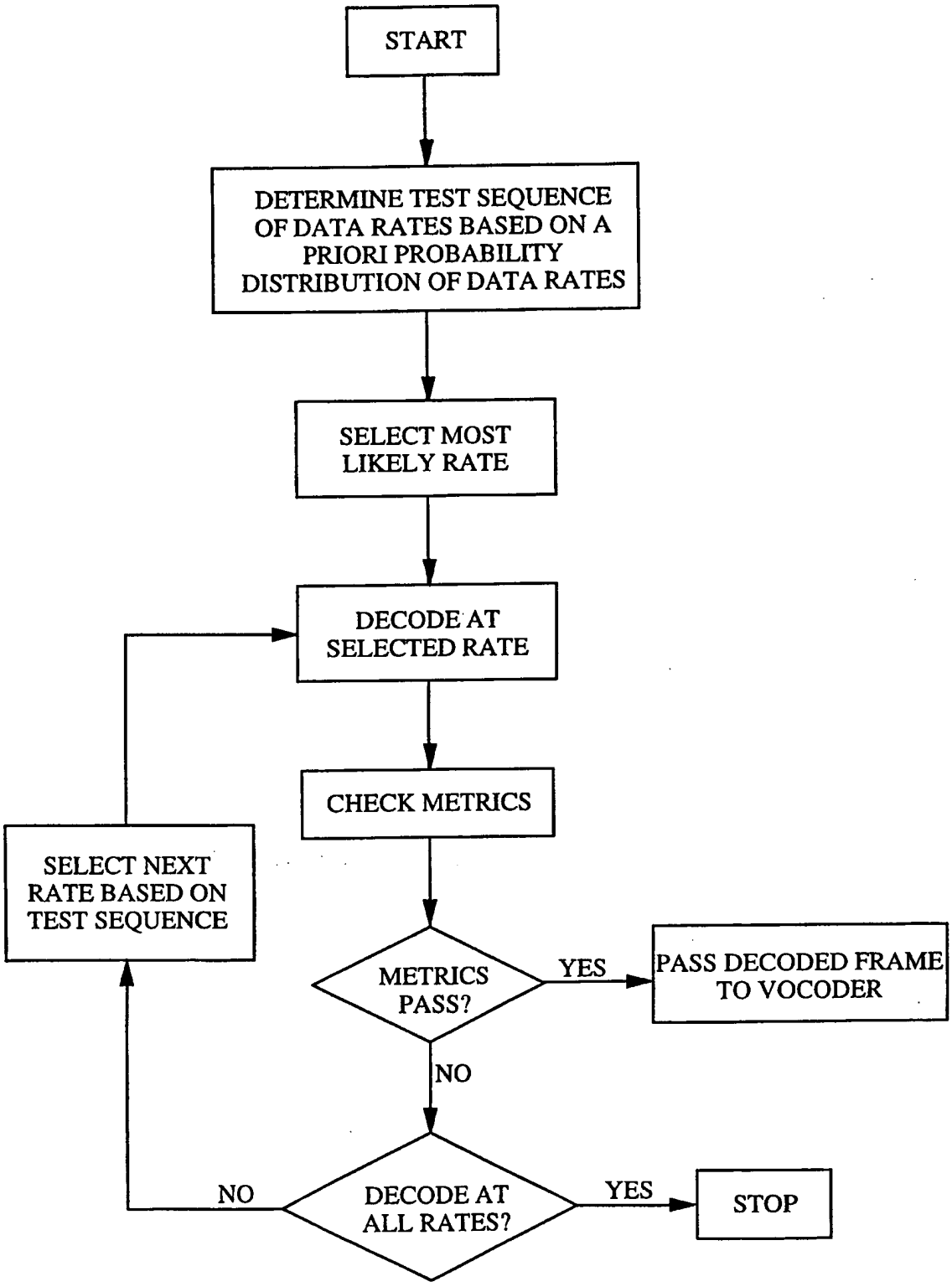

START
DETERMINE TEST SEQUENCE OF DATA RATES BASED ON A PRIORI PROBABILITY DISTRIBUTION OF DATA RATES
SELECT MOST LIKELY RATE
DECODE AT SELECTED RATE
CHECK METRICS
SELECT NEXT RATE BASED ON TEST SEQUENCE
METRICS PASS?
YES
PASS DECODED FRAME TO VOCODER
NO
DECODE AT ALL RATES?
NO
YES
STOP

FIG. 5

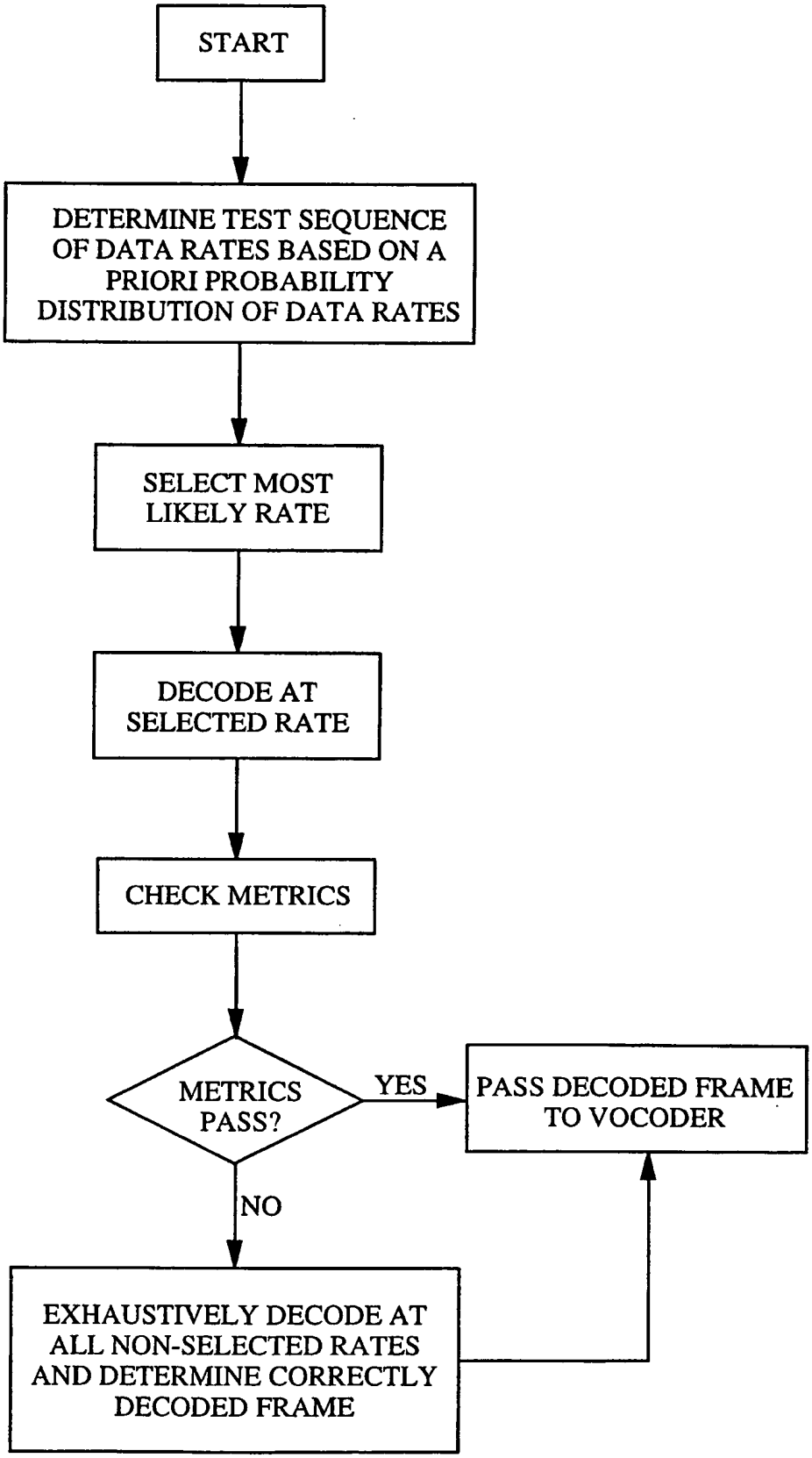

START
DETERMINE TEST SEQUENCE OF DATA RATES BASED ON A PRIORI PROBABILITY DISTRIBUTION OF DATA RATES
SELECT MOST LIKELY RATE
DECODE AT SELECTED RATE
CHECK METRICS
METRICS PASS?
YES
PASS DECODED FRAME TO VOCODER
NO
EXHAUSTIVELY DECODE AT ALL NON-SELECTED RATES AND DETERMINE CORRECTLY DECODED FRAME

FIG. 6

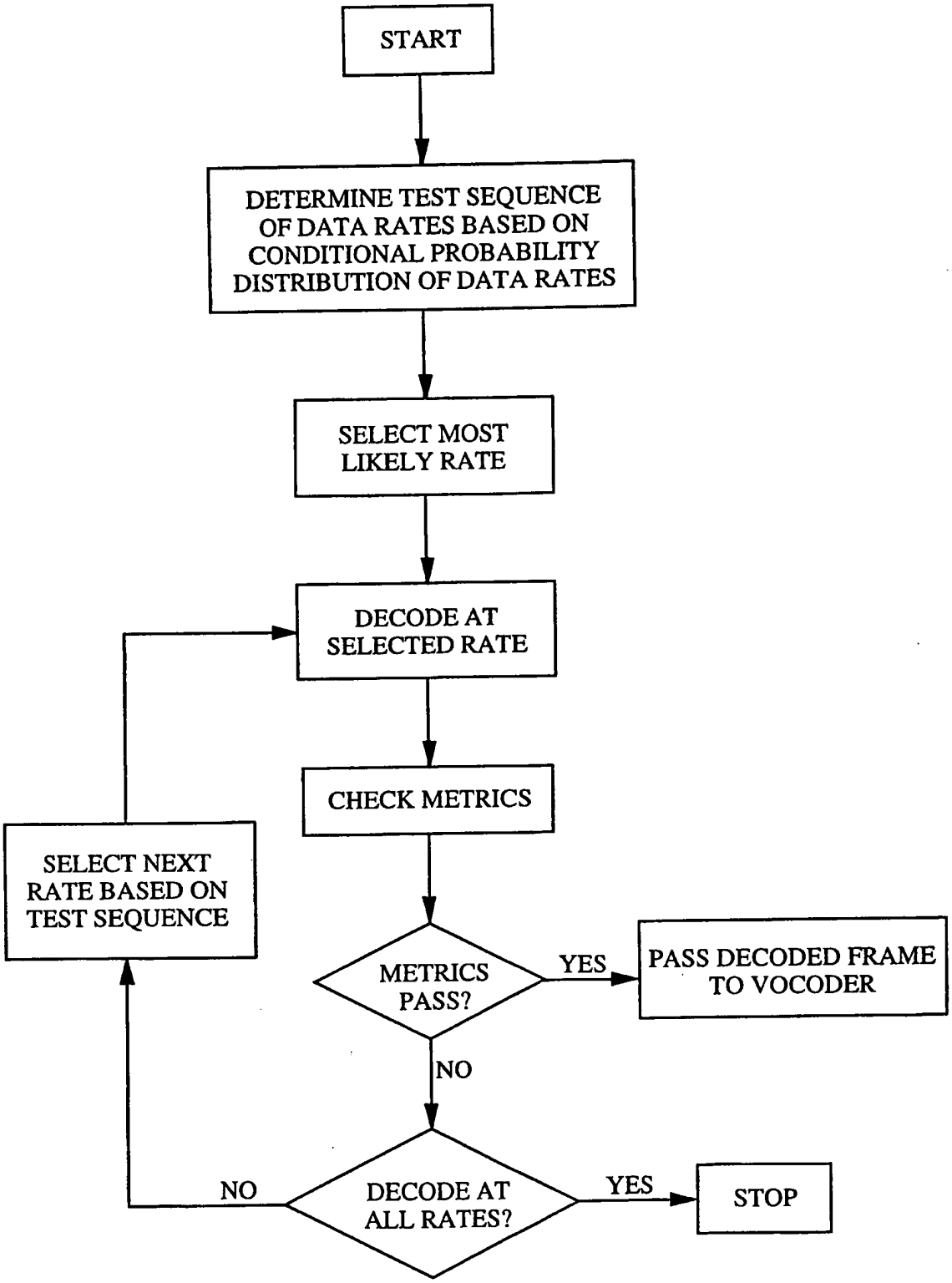

START
DETERMINE TEST SEQUENCE OF DATA RATES BASED ON CONDITIONAL PROBABILITY DISTRIBUTION OF DATA RATES
SELECT MOST LIKELY RATE
DECODE AT SELECTED RATE
CHECK METRICS
SELECT NEXT RATE BASED ON TEST SEQUENCE
METRICS PASS?
YES
PASS DECODED FRAME TO VOCODER
NO
NO
DECODE AT ALL RATES?
YES
STOP

FIG. 7

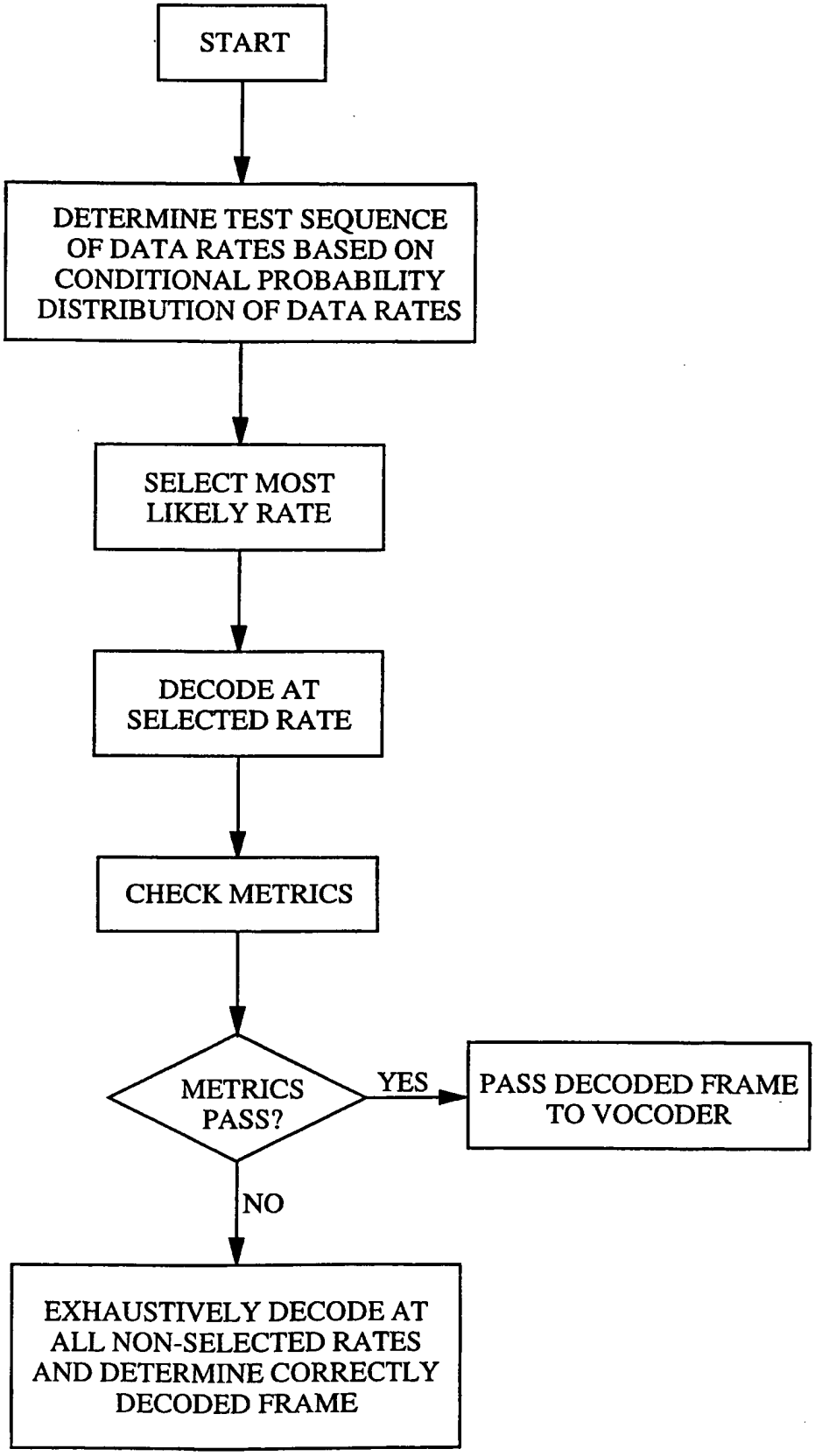

START
DETERMINE TEST SEQUENCE OF DATA RATES BASED ON CONDITIONAL PROBABILITY DISTRIBUTION OF DATA RATES
SELECT MOST LIKELY RATE
DECODE AT SELECTED RATE
CHECK METRICS
METRICS PASS?
YES
PASS DECODED FRAME TO VOCODER
NO
EXHAUSTIVELY DECODE AT ALL NON-SELECTED RATES AND DETERMINE CORRECTLY DECODED FRAME

FIG. 8

INTERNATIONAL SEARCH REPORT

Inte. ional Application No
PCT/US 97/19676

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 H04L25/02

According to International Patent Classification(IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 H04L

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	EP 0 711 056 A (NIPPON ELECTRIC CO) 8 May 1996 see abstract see page 2, column 2, line 46 - page 3, column 3, line 26 see page 4, column 5, line 13 - column 6, line 23; figure 2 ---	1,14,15, 22,27,28
A	EP 0 713 305 A (NIPPON ELECTRIC CO) 22 May 1996 see abstract see page 2, column 2, line 45 - page 3, column 3, line 12 see page 3, column 3, line 33 - column 4, line 2; figure 1 see page 3, column 4, line 44 - page 4, column 5, line 13 --- -/--	1,14,15, 22,27,28

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

1

Date of the actual completion of the international search	Date of mailing of the international search report
27 February 1998	09/03/1998

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Bossen, M

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Inte .onal Application No
PCT/US 97/19676

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
A	US 5 566 206 A (BUTLER BRIAN K ET AL) 15 October 1996 cited in the application see column 2, line 37 - line 47 see column 5, line 57 - column 6, line 32; figure 2 see column 7, line 35 - line 52; figure 4	7,9-12, 14, 22-25,27

1

Form PCT/ISA/210 (continuation of second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Information on patent family members

Int. ..ional Application No
PCT/US 97/19676

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
EP 0711056 A	08-05-96	JP 8130535 A	21-05-96
		AU 3662595 A	09-05-96
		CA 2163134 A	03-05-96
EP 0713305 A	22-05-96	JP 2596392 B	02-04-97
		JP 8149567 A	07-06-96
		AU 686026 B	29-01-98
		AU 3787595 A	23-05-96
		CA 2162417 A	17-05-96
		FI 955398 A	17-05-96
US 5566206 A	15-10-96	AT 158910 T	15-10-97
		AU 683479 B	13-11-97
		AU 7113694 A	17-01-95
		BR 9406891 A	26-03-96
		DE 69405997 D	06-11-97
		EP 0705512 A	10-04-96
		FI 956091 A	16-02-96
		JP 9501548 T	10-02-97
		WO 9501032 A	05-01-95
		CN 1108834 A	20-09-95
		IL 109842 A	30-09-97
		MX 9404610 A	31-01-95
		ZA 9404032 A	09-03-95

CORRECTED VERSION*

PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : H04L 9/00

A1

(11) International Publication Number: WO 98/11690

(43) International Publication Date: 19 March 1998 (19.03.98)

(21) International Application Number: PCT/US97/16223

(22) International Filing Date: 12 September 1997 (12.09.97)

(30) Priority Data:

60/025,991	12 September 1996 (12.09.96)	US
08/887,723	3 July 1997 (03.07.97)	US

(71)(72) Applicant and Inventor: GLOVER, John, J. [US/US]; 26 Amaranth Avenue, Medford, MA 02155 (US).

(74) Agent: GORDON, Peter, J.; Wolf, Greenfield & Sacks, P.C., 600 Atlantic Avenue, Boston, MA 02210 (US).

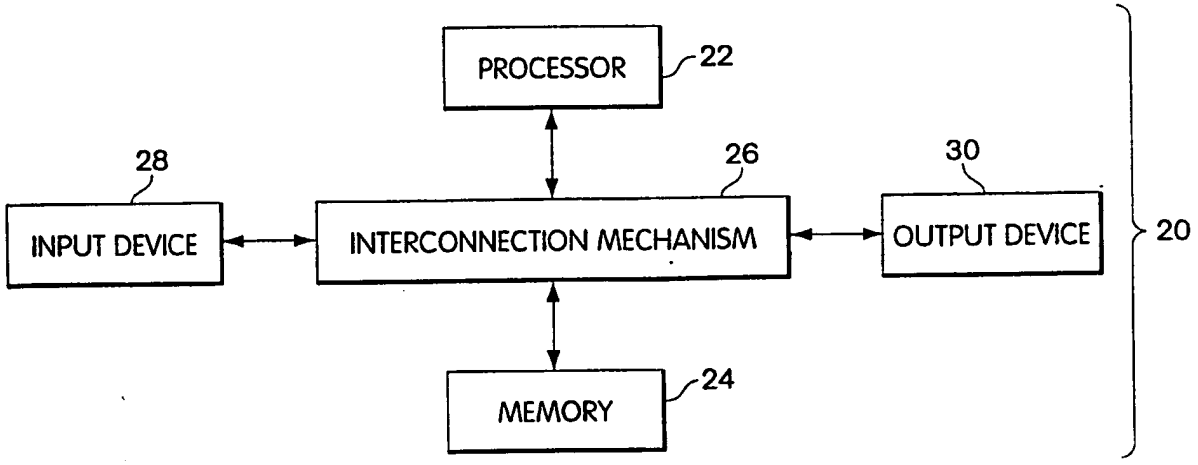
(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, HU, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW, ARIPO patent (GH, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published

With international search report.

Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: SELF-DECRYPTING DIGITAL INFORMATION SYSTEM AND METHOD



(57) Abstract

The claimed data protection device (20) includes a processor (22) connected to a memory system (24) through an interconnection mechanism (26). An input device (28) is also connected to the processor (22) and memory system (24) through the interconnection mechanism (26). The interconnection mechanism (26) is typically a combination of one or more buses and one or more switches. The output device (30) may be a display, and the input device (28) may be a keyboard and/or mouse or other cursor control device.

*(Referred to in PCT Gazette No. 32/1998, Section II)

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

SELF-DECRYPTING DIGITAL INFORMATION SYSTEM AND METHOD

Field of the Invention

The present invention is related to mechanisms for protecting digital information from being copied. In particular, the present invention is related to mechanisms which permit authorized execution of computer program code or access to other digital information which is encrypted or otherwise encoded.

Background of the Invention

A serious problem which faces the electronic publishing and software industries is the ease with which digital information can be copied without authorization from the publisher. Digital information also may be used or modified without authorization. For example, computer software may be reverse engineered or attacked by computer viruses.

There are many mechanisms available which may be used to limit or prevent access to digital information. Such mechanisms often either restrict the ability of the user to make back-up copies or involve the use of special purpose hardware to limit access to the digital information. For example, some mechanisms restrict the use of digital information to a particular machine. See, for example, U.S. Patent 4,817,140. Other mechanisms require the digital information to be stored on a particular recording medium in order to be used. See, for example, U.S. Patent 5,412,718. Yet other mechanisms allow only a certain number of uses of the digital information. See for example, U.S. Patent 4,888,798. Many of these access control mechanisms cause distribution to be more costly.

Several other patents describe a variety of systems for encryption, compression, licensing and royalty control and software distribution such as: U.S. Pat. No. 4,405,829, U.S. Pat. No. 4,864,616, U.S. Pat. No. 4,888,800, U.S. Pat. No. 4,999,806, U.S. Pat. No. 5,021,997, U.S. Patent No. 5,027,396, U.S. Pat. No. 5,033,084, U.S. Pat. No. 5,081,675, U.S. Pat. No. 5,155,847, U.S. Pat. No. 5,166,886, U.S. Pat. No. 5,191,611, U.S. Pat. No. 5,220,606, U.S. Pat. No. 5,222,133, U.S. Pat. No. 5,272,755, U.S. Pat. No. 5,287,407, U.S. Pat. No. 5,313,521, U.S. Pat. No. 5,325,433, U.S. Pat. No. 5,327,563, U.S. Pat. No. 5,337,357, U.S. Pat. No. 5,351,293, U.S. Pat. No. 5,341,429, U.S. Pat. No. 5,351,297, U.S. Pat. No. 5,361,359, U.S. Pat. No. 5,379,433, U.S. Pat. No. 5,392,351, U.S. Pat. No. 5,394,469, U.S. Pat. No. 5,414,850, U.S. Pat. No. 5,473,687, U.S. Pat. No. 5,490,216, U.S. Pat. No. 5,497,423, U.S. Pat. No. 5,509,074, U.S.

Pat. No. 5,,511,123, U.S. Pat. No. 5,524,072, U.S. Pat. No. 5,532,920, U.S. Pat. No. 5,555,304, U.S. Pat. No. 5,557,346, U.S. Pat. No. 5,557,765, U.S. Pat. No. 5,592,549, U.S. Pat. No. 5,615,264, U.S. Pat. No. 5,625,692, and U.S. Pat. No. 5,638,445.

Computer programs or other digital information also may be encrypted in order to prevent an individual from making a useful copy of the information or from reverse engineering a program. Even with such encryption, however, a computer program must be decrypted in order for a computer to load and execute the program. Similarly, other digital information must be decrypted before it can be accessed and used. Generally, digital information is decrypted to disk, and not to main memory of the computer which is more protected by the operating system, because decryption to main memory results in a significant loss of memory resources. If the purpose for using encryption is to prevent users from copying the digital information, then decryption of the information to accessible memory for use defeats this purpose.

One way to protect digital information using encryption has been made available by International Business Machines (IBM) and is called a "CRYPTOLOPE" information container. This technology is believed to be related to U.S. Patent Nos. 5,563,946 and 5,598,470 (to Cooper et al.), and published European patent applications 0679977, 0679978, 0679979 and 0681233. The CRYPTOLOPE system requires a user to have a "helper application" and a key. The CRYPTOLOPE information container is generated by IBM. The content provider submits data to IBM, which in turn encrypts and packages the data in a CRYPTOLOPE information container. The helper application is a form of memory resident program, called a terminate and stay resident (TSR) program, which is a form of input/output (I/O) device driver installed in the operating system and which monitors requests from the operating system for files on specified drives and directories. Because the TSR program must know the directory, and/or file name to be accessed, that information also is available to other programs. Other programs could use that information to manipulate the operation of the TSR program in order to have access to decrypted contents of the information container. The encrypted information container includes an executable stub which is executed whenever the application is run without the installed TSR program or from a drive not monitored by the TSR program to prevent unpredictable activity from executing encrypted code. This stub may be used to install decryption and cause the application be executed a second time, or to communicate with the TSR program to instruct the TSR program to monitor the drive. It may be preferable from the point of view of the content provider however to maintain an encryption process and keys independently of any third party.

Multimedia content, such as a movie or hypertext presentation also may be stored on a digital versatile disk (DVD), sometimes called a digital video disk, compact disk read-only memory (CD-ROM), rewriteable compact disks (CD-RW) or other medium in an encrypted digital format for use with special-purpose devices. For example, concern about illegal copying of content from digital video disks or other digital media has resulted in a limited amount of content being available for such devices. This problem has caused representatives of both multimedia providers and digital video disk manufacturers to negotiate an agreement on an encryption format for information stored on DVDs. This copy protection scheme is licensed through an organization called the CSS Interim Licensing organization. However, in this arrangement, the content provider is limited to using the agreed upon encryption format and a device manufacturer is limited to using a predetermined decryption system.

Encryption has also been used to protect and hide computer viruses. Such viruses are typically polymorphic, i.e., they change every time they infect a new program, and are encrypted. The virus includes a decryption program that executes to decrypt the virus every time the infected program is run. Such viruses are described, for example, in "Computer Virus-Antivirus Coevolution" by Carey Nachenberg, Communications of the ACM, Vol. 40, No. 1, (Jan. 1997), p. 46 et seq. Such viruses include decryption keys within them since, clearly, their execution is not carried out by the user and a user would not be asked for authorization keys to permit execution of the viruses. Additionally, such viruses are typically only executed once at the start of execution of an infected program and permanently return control to the infected program after execution.

Summary of the Invention

Some of these problems with digital information protection systems may be overcome by providing a mechanism which allows a content provider to encrypt digital information without requiring either a hardware or platform manufacturer or a content consumer to provide support for the specific form of corresponding decryption. This mechanism can be provided in a manner which allows the digital information to be copied easily for back-up purposes and to be transferred easily for distribution, but which should not permit copying of the digital information in decrypted form. In particular, the encrypted digital information is stored as an executable computer program which includes a decryption program that decrypts the encrypted information

to provide the desired digital information, upon successful completion of an authorization procedure by the user.

In one embodiment, the decryption program is executed as a process within a given operating system and decrypts the digital information within the memory area assigned to that process. This memory area is protected by the operating system from copying or access by other processes. Even if access to the memory area could be obtained, for example through the operating system, when the digital information is a very large application program or a large data file, a copy of the entire decrypted digital information is not likely to exist in the memory area in complete form.

By encrypting information in this manner, a platform provider merely provides a computer system with an operating system that has adequate security to define a protected memory area for a process and adequate functionality to execute a decryption program. The content provider in turn may use any desired encryption program. In addition, by having a process decrypt information within a protected memory area provided by the operating system, the decrypted information does not pass through any device driver, memory resident program or other known logical entity in the computer system whose behavior may be controlled to provide unauthorized access to the data. The ability to reverse engineer or attack a computer program with a computer virus also may be reduced.

In another embodiment, the decryption program is part of a dynamically loaded device driver that responds to requests for data from the file containing the encrypted data. When the digital information product is first executed, this device driver is extracted from the file and is loaded into the operating system. The executed digital information product then informs the loaded device driver of the location of the hidden information in the file, any keys or other passwords, and the name of a phantom directory and file to be called that only the digital information product and the device driver know about. The name of this directory may be generated randomly. Each segment of hidden information in the digital information product may be assigned its own unique file name in the phantom directory. The digital information product then makes a call to the operating system to execute one of the files in the phantom directory. The loaded driver traps these calls to the operating system, accesses the original file, decrypts the desired information and outputs the desired information to the operating system.

In combination with other mechanisms that track distribution, enforce royalty payments and control access to decryption keys, the present invention provides an improved method for

identifying and detecting sources of unauthorized copies. Suitable authorization procedures also enable the digital information to be distributed for a limited number of uses and/or users, thus enabling per-use fees to be charged for the digital information.

Accordingly, one aspect of the invention is a digital information product including a computer-readable medium with digital information stored thereon. The digital information includes computer program logic having a first portion of executable computer program logic and a second portion of digital information. The first portion of executable program logic, when executed, defines a mechanism for responding to requests for digital information from an operating system of a computer. This mechanism, when used to access the second portion of the encrypted digital information, decrypts the encrypted digital information, and provides the encrypted digital information to the operating system.

In the foregoing aspect of the invention, the digital information may be executable computer program logic. Hence, one aspect of the invention is a computer program product, including a computer readable medium with computer program logic stored thereon. The computer program logic includes a first portion of executable computer program logic and a second portion of encrypted computer program logic. The first portion of executable computer program logic, when executed, defines a mechanism for responding to requests for computer program logic from an operating system of a computer. This mechanism accesses the second portion of encrypted computer program logic, decrypts the encrypted computer program logic, and provides the decrypted computer program logic to the operating system.

Another aspect of the present invention is a computer program product, a computer system and a process which produce a computer program or digital information product in accordance with other aspects of the invention, using executable program code for the first and second portions of the desired computer program product.

Another aspect of the present invention is a computer program product including a self-decrypting encrypted executable computer program. The product includes a computer readable medium having computer program logic stored thereon. The computer program logic defines first, second and third modules, wherein the third module defines the encrypted executable computer program. The first module, when executed by a computer, defines a mechanism for loading the second module into memory of the computer. The second module, when executed by a computer, defines a mechanism for communicating with an operating system of the computer to receive requests for program code from the encrypted executable computer program from the

third module, and for processing the requests to access and decrypt the encrypted executable computer program and for providing the decrypted executable code from the third module to the operating system.

Another aspect of the invention is a process for executing encrypted executable computer programs on a computer system having a processor, memory and operating system. The process involves receiving computer program logic having a first module defining a start up routine, a second module, and a third module containing the encrypted executable computer program. The first module of the received computer program logic is executed using the processor. When the first module is executed, the second module is caused to be loaded into the memory of the computer system. Requests are generated from the operating system for data from the encrypted executable computer program and are received by the second module. The second module accesses and decrypts the encrypted executable computer program in response to these requests and returns the decrypted executable computer program to the operating system.

These and other aspects, advantages and features of the present invention and its embodiments will be more apparent given the following detailed description.

Brief Description of the Drawing

In the drawing,

Fig. 1 is a block diagram of a typical computer system with which the present invention may be implemented;

Fig. 2 is a block diagram of a memory system in the computer system of Fig. 1;

Fig. 3 is a diagram of a computer program or digital information product which may be recorded on a computer readable and writable medium, such as a magnetic disc;

Fig. 4 is a flowchart describing how the computer program or digital information product of Fig. 3 is used;

Fig. 5 is a flowchart describing operation of an example unwrap procedure as shown in Fig. 3 in one embodiment of the invention;

Fig. 6 is a flowchart describing operation of an example device driver as shown in Fig. 3 in one embodiment of the invention;

Fig. 7 is a block diagram of a computer system in the process of executing a computer program product in accordance with one embodiment of the invention;

Fig. 8 is a flowchart describing operation of an example unwrap procedure in another embodiment of the invention; and

Fig. 9 is a flowchart describing how a computer program product such as shown in Fig. 3 is constructed.

Detailed Description

The present invention will be more completely understood through the following detailed description which should be read in conjunction with the attached drawing in which similar reference numbers indicate similar structures.

Embodiments of the present invention may be implemented using a general purpose digital computer or may be implemented for use with a digital computer or digital processing circuit. A typical computer system 20 is shown in Fig. 1, and includes a processor 22 connected to a memory system 24 via an interconnection mechanism 26. An input device 28 also is connected to the processor and memory system via the interconnection mechanism, as is an output device 30. The interconnection mechanism 26 is typically a combination of one or more buses and one or more switches. The output device 30 may be a display and the input device may be a keyboard and/or a mouse or other cursor control device.

It should be understood that one or more output devices 30 may be connected to the computer system. Example output devices include a cathode ray tube (CRT) display, liquid crystal display (LCD), television signal encoder for connection to a television or video tape recorder, printers, communication devices, such as a modem, and audio output. It also should be understood that one or more input devices 28 may be connected to the computer system. Example input devices include a keyboard, keypad, trackball, mouse, pen and tablet, communication device, audio or video input and scanner. It should be understood that the invention is not limited to the particular input or output devices used in combination with the computer system or to those described herein.

The computer system 20 may be a general purpose computer system, which is programmable using a high level computer programming language, such as "C++," "Pascal," "VisualBasic." The computer system also may be implemented using specially programmed, special purpose hardware. In a general purpose computer system, the processor is typically a commercially available processor, such as the Pentium processor from Intel Corporation. Many other processors are also available. Such a processor executes a program called an operating

system, such as Windows 95 or Windows NT 4.0, both available from Microsoft Corporation, which controls the execution of other computer programs and provides scheduling, debugging, input output control, accounting compilation, storage assignment, data management and memory management, and communication control and related services. Other examples of operating systems include: MacOS System 7 from Apple Computer, OS/2 from IBM, VMS from Digital Equipment Corporation, MS-DOS from Microsoft Corporation, UNIX from AT&T, and IRIX from Silicon Graphics, Inc.

The computer system 20 also may be a special purpose computer system such as a digital versatile disk or digital video disk (DVD) player. In a DVD player, there is typically a decoder controlled by some general processor which decodes an incoming stream of data from a DVD. In some instances, the DVD player includes a highly integrated DVD decoder engine. Such devices generally have a simple operating system which may be modified to include the capabilities described and used herein in connection with the typical operating systems in a general purpose computer. In particular, some operating systems are designed to be small enough for installation in an embedded system such as a DVD player, including the WindowsCE operating system from Microsoft Corporation and the JavaOS operating system from SunSoft Corporation. The operating system allows a content provider to provide its own programs that define some of the content, which is particularly useful for interactive multimedia. This capability also can be used to provide encryption and decryption, in accordance with the invention.

The processor and operating system define a computer platform for which application programs in a programming language such as an assembly language or a high level programming language are written. It should be understood that the invention is not limited to a particular computer platform, operating system, processor, or programming language. Additionally, the computer system 20 may be a multi-processor computer system or may include multiple computers connected over a computer network.

An example memory system 24 will now be described in more detail in connection with Fig. 2. A memory system typically includes a computer readable and writable non-volatile recording medium 40, of which a magnetic disk, a flash memory, rewriteable compact disk (CD-RW) and tape are examples. The recording medium 40 also may be a read only medium such as a compact disc-read only memory (CD-ROM) or DVD. A magnetic disk may be removable, such as a "floppy disk" or "optical disk," and/or permanent, such as a "hard drive." The disk,

which is shown in Fig. 2, has a number of tracks, as indicated at 42, in which signals are stored, in binary form, i.e., a form interpreted as a sequence of 1's and 0's, as shown at 44. Such signals may define an application program to be executed by the microprocessor, or information stored on the disk to be processed by the application program. Typically, in the operation of a general purpose computer, the processor 22 causes data to be read from the non-volatile recording medium 40 into an integrated circuit memory element 46, which is typically a volatile random access memory, such as a dynamic random access memory (DRAM) or static random access memory (SRAM). The integrated circuit memory element 46 allows for faster access to the information by the processor than disk 40, and is typically called the system or host memory. The processor generally causes the data to be manipulated within the integrated circuit memory 46 and may copy the data to the disk 40, if modified, when processing is completed. A variety of mechanisms are known for managing data movement between the disk 40 and the integrated circuit memory 46, and the invention is not limited thereto. It should also be understood that the invention is not limited to a particular memory system.

The file system of a computer generally is the mechanism by which an operating system manages manipulation of data between primary and secondary storage, using files. A file is a named logical construct which is defined and implemented by the operating system to map the name and a sequence of logical records of data to physical storage media. An operating system may specifically support various record types or may leave them undefined to be interpreted or controlled by application programs. A file is referred to by its name by application programs and is accessed through the operating system using commands defined by the operating system. An operating system provides basic file operations provided by for creating a file, opening a file, writing a file, reading a file and closing a file.

In order to create a file, the operating system first identifies space in the storage media which is controlled by the file system. An entry for the new file is then made in a directory which includes entries indicating the names of the available files and their locations in the file system. Creation of a file may include allocating certain available space to the file. Opening a file returns a handle to the application program which it uses to access the file. Closing a file invalidates the handle.

In order to write data to a file, an application program issues a command to the operating system which specifies both an indicator of the file, such as a file name, handle or other descriptor, and the information to be written to the file. Given the indicator of the file, the

operating system searches the directory to find the location of the file. The directory entry stores a pointer, called the write pointer, to the current end of the file. Using this pointer, the physical location of the next available block of storage is computed and the information is written to that block. The write pointer is updated in the directory to indicate the new end of the file.

In order to read data from a file, an application program issues a command to the operating system specifying the indicator of the file and the memory locations assigned to the application where the next block of data should be placed. The operating system searches its directory for the associated entry given the indicator of the file. The directory may provide a pointer to a next block of data to be read, or the application may program or specify some offset from the beginning of the file to be used.

A primary advantage of using a file system is that, for an application program, the file is a logical construct which can be created, opened, written to, read from and closed without any concern for the physical storage used by the operating system.

The operating system also allows for the definition of another logical construct called a process. A process is a program in execution. Each process, depending on the operating system, generally has a process identifier and is represented in an operating system by a data structure which includes information associated with the process, such as the state of the process, a program counter indicating the address of the next instruction to be executed for the process, other registers used by process and memory management information including base and bounds registers. Other information also may be provided. The base and bounds registers specified for a process contain values representing the largest and smallest addresses that can be generated and accessed by an individual program. Where an operating system is the sole entity able to modify these memory management registers, adequate protection from access to the memory locations of one process from another process is provided. As a result, this memory management information is used by the operating system to provide a protected memory area for the process. A process generally uses the file system of the operating system to access files.

The present invention involves storing encrypted digital information, such an audio, video, text or an executable computer program, on a computer readable medium such that it can be copied easily for back-up purposes and transferred easily for distribution, but also such that it cannot be copied readily in decrypted form during use. In particular, the digital information is stored as a computer program that decrypts itself while it is used to provide the digital information, e.g., to provide executable operation code to the operating system of a computer, as

the digital information is needed. Any kind of encryption or decryption may be used and also may include authorization mechanisms and data compression and decompression. In one embodiment of the present invention, decrypted digital information exists only in memory accessible to the operating system and processes authorized by the operating system. When the digital information is a large application program, a copy of the entire decrypted application program is not likely to exist in the main memory at any given time, further reducing the likelihood that a useful copy of decrypted code could be made. The decryption operation also is performed only if some predetermined authorization procedure is completed successfully.

One embodiment of the invention, in which the decryption program is a form of dynamically loaded device driver, will first be described. Fig. 3 illustrates the structure of digital information as stored in accordance with one embodiment of the present invention, which may be stored on a computer readable medium such as a magnetic disc or compact disc read only memory (CD-ROM) to form a computer program product. The digital information includes a first portion 50, herein called an unwrap procedure or application, which is generally unencrypted executable program code. The purpose of the unwrap procedure is to identify the locations of the other portions of the digital information, and may perform other operations such as verification. In particular, the unwrap procedure identifies and extracts a program which will communicate with the operating system, herein called a virtual device driver 52. The unwrap procedure may include decryption and decompression procedures to enable it to decrypt/decompress the driver, and/or other content of this file. The program 52 need not be a device driver. The virtual device driver 52 typically follows the unwrap procedure 50 in the file container, the digital information. The virtual device driver, when executed, decrypts and decodes the desired digital information such as an executable computer program code from hidden information 54, which may be either encrypted and/or encoded (compressed). It is the decrypted hidden information which is the desired digital information to be accessed. This hidden information may be any kind of digital data, such as audio, video, text, and computer program code including linked libraries or other device drivers.

In this embodiment of the computer program product, labels delineate the boundaries between the device driver and the hidden files. These labels may or may not be encrypted. A first label 56 indicates the beginning of the code for the virtual device driver 52. A second label 58 indicates the end of the virtual device driver code. Another label 60 indicates the beginning of the hidden information and a label 62 indicates the end of that application. There may be one

or more blocks of such hidden information, each of which can be given a different name. It may be advantageous to use the name of the block of information in its begin and end tags. This computer program product thus contains and is both executable computer program code and one or more blocks of digital information. A table of locations specifying the location of each portion of the product could be used instead of labels. Such a table could be stored in a predetermined location and also may be encrypted.

The overall process performed using this computer program product in one embodiment of the invention will now be described in connection with Fig. 4. This embodiment may be implemented for use with the Windows95 operating system and is described in more detail in connection with Figs. 5-7. An embodiment which may be implemented for use on the WindowsNT 4.0 operating system is described in more detail below in connection with Fig. 8. In both of these described embodiments, the digital information is an executable computer program which is read by the operating system as data from this file and is executed. The same principle of operation would apply if the data were merely audio, video, text or other information to be conveyed by a user. In the embodiment of Fig. 4, the computer program is first loaded into memory in step 70, and the unwrap procedure 50 is executed by the operating system, as any typical executable computer program is executed. The unwrap procedure may perform authorization, for example by checking for a required password or authentication code, and may receive any data needed for decryption or decompression, for example keys or passwords, in step 72. Suitable authorization procedures may provide the ability to distribute software for single use. The unwrap procedure locates the virtual device driver 52 within the computer program in step 74, and then locates the hidden application in step 76. The virtual device driver 52 is then extracted by the unwrap procedure from the computer program, copied to another memory location and loaded for use by the operating system in step 78. An advantage of an operating system like Windows95 is that it allows such device drivers to be loaded dynamically without restarting the computer.

The executed unwrap procedure 50, in step 80, informs the loaded virtual device driver 52 of the location of the hidden information in the file, any keys or other passwords, and a name of a phantom directory and file to be called that only the unwrap procedure and the virtual device driver know about. The name of this phantom directory may be generated randomly. Each segment information hidden in the digital information product may be assigned its own unique file name in the phantom directory.

After the loaded virtual device driver 52 receives all communications from the unwrap procedure, it opens the original application file for read only access in step 82. The unwrap procedure then makes a call to the operating system in step 84 to execute the file in the phantom directory for which the name was transmitted to the loaded virtual device driver. One function of the loaded virtual device driver 52 is to trap all calls from the operating system to access files in step 86. Any calls made by the operating system to access files in the phantom directory are processed by the virtual device driver, whereas calls to access files in other directories are allowed to proceed to their original destination. In response to each call from the operating system, the virtual device driver obtains the bytes of data requested by the operating system from the original computer program file in step 88. These bytes of data are then decrypted or decompressed in step 90 and returned to the operating system. When processing is complete, the phantom application is unloaded from the operating system in step 92, and may be deleted from the memory.

A more detailed description of the process of Fig. 4 will now be described in connection with Figs. 5-7. Fig. 5 is a flowchart describing the operation of one embodiment of the unwrap procedure in more detail. The first step performed by this procedure is identifying the operating system being used, in step 100. This step is useful because different methods may be used with different operating systems. All code that may be used to run in various operating systems may be placed in this unwrap procedure. This procedure also may contain the decompression/decryption code, for example or any other computer program code to be executed.

The executed application then opens the original executable file as a data file and searches for the begin and end tags of the device driver and hidden files in step 102. The device driver code is copied into memory and loaded into the operating system in step 104. The unwrap procedure then informs the device driver of the name of the original application file, offsets of the hidden files and the name of a phantom directory, which is typically randomly generated (step 106). This communication may be performed using a "DeviceIOControl" function call in the Windows95 operating system. The unwrap procedure then makes a call to the operating system to execute the hidden file in the phantom directory, in step 108.

The operation of one embodiment of a device driver will now be described in connection with Fig. 6. After the device driver is loaded into the operating system, it hooks into a position between the operating system and a file system driver (FSD), in step 110, to intercept

calls made by the operating system to the FSD for data from files in the phantom directory. The FSD is the code within the operating system that performs physical reading and writing of data to disk drives. The operating system makes requests to the FSD for data from files in directories on the disk drives. The driver then receives information from the unwrap procedure including the name of the original file, the location of hidden files within the original file, and the name of the phantom directory created by the unwrap procedure (step 112). The device driver opens the original file as a read only data file. The device driver now traps calls, in step 114, made from the operating system for files in the phantom directory. Calls to other directories are ignored and passed on to the original destination. The device driver then reads the data from the original data file, decrypts and decompresses it, and returns the decrypted/decompressed data to the operating system in step 116.

For example, if the offset for the hidden application in the original data file is 266,270 bytes and the operating system asks for 64 bytes starting at offset 0 of the hidden application in the phantom directory, the device driver reads 64 bytes from the original file starting at offset 266,270, decrypts/decompresses those 64 bytes, and returns the first 64 decrypted/decompressed bytes back to the operating system. From the point of view of the operating system, the 64 bytes appear to have come from the file in the phantom directory. Steps 114 and 116 are performed on demand in response to the operating system.

A block diagram of the computer system in this embodiment, with a device driver loaded and in operation, will now be described in more detail in connection with Fig. 7. Fig. 7 illustrates the operating system 120, the loaded device driver 122, a file system driver 124, the original executable file 126 as it may appear on disk and the unwrap procedure 128. The executable file may in fact be on a remote computer and accessed through a network by the device driver. The unwrap procedure causes the operating system to begin execution of the hidden file by issuing an instruction to execute the file in the phantom directory, as indicated at 130. This command is issued after the device driver 122 is informed of the file name of the original executable file 126, offsets of the hidden files within that file and the name of the phantom directory, as indicated at 132. The operating system then starts making calls to the phantom directory as indicated at 134. The device driver 122 traps these calls and turns them into requests 136 to the file system driver to access the original executable file 126. Such requests actually are made to the operating system 120, through the device driver 122 to the file system driver 124. The file system driver 124 returns encrypted code 138 to the device driver

122. The encrypted code 138 actually passes back through the device driver 122 to the operating system 120 which in turn provides the encrypted code 138 to the device driver 122 as the reply to the request 136 for the original file. The device driver 122 then decrypts the code to provide decrypted code 140 to the operating system 120.

Another embodiment of the invention will now be described in connection with Fig. 8. This embodiment may be implemented using the WindowsNT 4.0 operating system, for example. In this embodiment, the device driver portion 52 of the computer program product is not used. The unwrap procedure for this embodiment begins by identifying the operating system being used similar, which is step 100 in Fig. 5. If the operating system is Windows NT 4.0, for example, a different unwrap procedure for this embodiment is performed. Before describing this unwrap procedure, a brief description of some of the available operating system commands will be provided.

Currently, under all versions of the Window operating system or operating environment from Microsoft Corporation (such as Windows 3.1, Windows 95 and Windows NT 3.51 and 4.0) all executable files (.exe) or dynamic link library (.dll and .ocx) files, which are executable files with different header and loading requirements than .exe files, that are loaded into memory by the operating system must reside as a file either locally, e.g., on a disk drive or remotely, e.g., over a network or communications port. All further references herein to loading an executable will be using the Win32 function calls used in Windows 95 and NT 3.51 and 4.0 operating systems. The CreateProcess() function which loads files with an .exe extension takes ten parameters:

```
BOOL CreateProcess(// Prototype from Microsoft Visual C++ Help Documentation
  LPCTSTR lpApplicationName,                  // pointer to name of executable module
  LPTSTR lpCommandLine,                       // pointer to command line string
  LPSECURITY_ATTRIBUTES lpProcessAttributes,  // pointer to process security attributes
  LPSECURITY_ATTRIBUTES lpThreadAttributes,   // pointer to thread security attributes
  BOOL bInheritHandles,                       // handle inheritance flag
  DWORD dwCreationFlags,                      // creation flags
  LPVOID lpEnvironment,                       // pointer to new environment block
  LPCTSTR lpCurrentDirectory,                 // pointer to current directory name
  LPSTARTUPINFO lpStartupInfo,                // pointer to STARTUPINFO
  LPPROCESS_INFORMATION lpProcessInformation  // pointer to PROCESS_INFORMATION
);
```

Three of these parameters are pointers to strings that contain an application file name, command line parameters, and the current directory. The other parameters are security, environmental, and process information. The LoadLibrary() function takes one parameter that is a pointer to a string that contains the application file name:

```
HINSTANCE LoadLibrary(// Prototype from Microsoft Visual C++ Help Documentation
 LPCTSTR lpLibFileName      // address of filename of executable module
);
```

The LoadLibraryEx() function takes three parameters the first being the same as LoadLibrary(), the second parameter must be null, and the third tells the operating system whether to load the file as an executable or as a data file in order to retrieve resources such as icons or string table data from it and not load it as an executable:

```
HINSTANCE LoadLibraryEx(// Prototype from Microsoft Visual C++ Help Documentation
 LPCTSTR lpLibFileName,     // points to name of executable module
 HANDLE hFile,              // reserved, must be NULL
 DWORD dwFlags              // entry-point execution flag
);
```

The CreateFile() function is used to create and open files and to load files such as device drivers. This function also requires a pointer to a string that contains the name of a physical file:

```
HANDLE CreateFile(// Prototype from Microsoft Visual C++ Help Documentation
 LPCTSTR lpFileName,                          // pointer to name of the file
 DWORD dwDesiredAccess,                       // access (read-write) mode
 DWORD dwShareMode,                           // share mode
 LPSECURITY_ATTRIBUTES lpSecurityAttributes,  // pointer to security descriptor
 DWORD dwCreationDistribution,                // how to create
 DWORD dwFlagsAndAttributes,                  // file attributes
 HANDLE hTemplateFile                         // handle to file with attributes to copy
);
```

There are other functions such as MapViewOfFile() and MapViewOfFileEx() that map areas of memory to an already opened physical file through a handle to that file. They have the following parameters:

```
LPVOID MapViewOfFile(// Prototype from Microsoft Visual C++ Help Documentation
  HANDLE hFileMappingObject,        // file-mapping object to map into address space
  DWORD dwDesiredAccess,            // access mode
  DWORD dwFileOffsetHigh,           // high-order 32 bits of file offset
  DWORD dwFileOffsetLow,            // low-order 32 bits of file offset
  DWORD dwNumberOfBytesToMap        // number of bytes to map
 );

LPVOID MapViewOfFileEx(// Prototype from Microsoft Visual C++ Help Documentation
  HANDLE hFileMappingObject,        // file-mapping object to map into address space
  DWORD dwDesiredAccess,            // access mode
  DWORD dwFileOffsetHigh,           // high-order 32 bits of file offset
  DWORD dwFileOffsetLow,            // low-order 32 bits of file offset
  DWORD dwNumberOfBytesToMap,       // number of bytes to map
  LPVOID lpBaseAddress              // suggested starting address for mapped view
 );
```

All of the foregoing functions directly use a pointer to a string that is a physical file. The only file functions that do not directly use a physical filename are functions like CreateNamedPipe(), which has the following parameters:

```
HANDLE CreateNamedPipe(// Prototype from Microsoft Visual C++ Help Documentation
  LPCTSTR lpName,                          // pointer to pipe name
  DWORD dwOpenMode,                        // pipe open mode
  DWORD dwPipeMode,                        // pipe-specific modes
  DWORD nMaxInstances,                     // maximum number of instances
  DWORD nOutBufferSize,                    // output buffer size, in bytes
  DWORD nInBufferSize,                     // input buffer size, in bytes
  DWORD nDefaultTimeOut,                   // time-out time, in milliseconds
  LPSECURITY_ATTRIBUTES lpSecurityAttributes  // pointer to security attributes structure
 );
```

The string to which CreateNamedPipe() points using the first parameter is a string that both an existing executable and the operating system know about and does not exist physically. Unfortunately both of the executables that "know" this private name could only be loaded using one of the other procedures that required a physical file. Currently it is not possible to load an executable using a "named pipe" name. Both of or any executables that use the name of the "named pipe" already must have been loaded into memory.

All of the foregoing functions require a physical file because all of them use "file mapping" processes. File mapping allows large executable files to appear to be loaded rapidly since they are rarely completely loaded into memory but rather are mapped into memory. The detriment to this mapping capability is that executable code must remain in physical memory in a file in unencrypted form in order to be loaded, unless there is a middle layer or file system driver that the operating system uses as a physical layer and that decrypts the executable code to the operating system on demand. The potential weakness here is that another file system driver can hook into the operating system to monitor traffic between the operating system and all file system drivers and capture decrypted executable code passing from the file system driver to the operating system. Some operating systems allow such monitoring more than others. Many anti-viral software packages use this technique to prevent computer virus attacks.

One method of loading and executing encrypted executable computer program code is to use a stub executable having two parts. The first part is the normal front end loader code that all executables have. In addition, the first part would perform any authorization which may include receiving a password from the user, then allocate enough memory to hold hidden encrypted code when it is decrypted, either in its entirety or a portion of it, copy the encrypted code into that area of protected (and preferably locked so no disk swapping occurs) memory, decrypt it once it is in memory and only in memory, and then have the operating system load the code only from memory therefore bypassing any file system drivers or TSRs so they have access to only encrypted code.

Some of the file functions listed above and similar functions on other operating systems could be modified easily by a programmer having access to source code for those operating systems, or a new operating system may be made to provide functions which allow direct loading of executable code from memory rather than physical files. For example, in the Win32 commands, a command similar to CreateProcess() command could be provided. The command should have a few extra parameters including the process identifier of the process that contains

the now decrypted executable code, the memory address of the start of the decrypted code, and the size of the decrypted code. The command could also contain a parameter specifying a "call back" function within the first process that would provide decrypted code on demand directly to the operating system through a protected buffer, therefore allowing only a portion of the encrypted code to be decrypted at any one time instead of in its entirety, for better protection and less memory use. The second parameter of the LoadLibraryEx() command that now needs to be NULL could be expanded to hold a structure that contained the same information. Both of these and other similar functions could be changed or created to allow loading executable code either as an .exe, .dll, or other extensions or identifiers, such as by using a "named pipe" name that only the operating system and process that holds decrypted code know about and having the operating system load from the named pipe.

Alternatively, without having such additional capabilities in the operating system, an application program can be divided into two parts. The first part is code that is common to all applications such as code for allocating memory off the heap and code that provides some interaction with the user. This kind of code is generally not code that the content provider is concerned about copying. The second part is the code that the content provider believes is valuable. Typically this valuable code is a business logic code or what would be considered a middle tier of a three-tier environment. A content provider would like to protect this second part of the code, at least much more that the first part of the code. The content provider would place all of the important code to be protected inside a dynamic link library and the code that is not that important would reside in the front end "stub" executable. Both of these would be combined into another executable containing the .dll in encrypted form only, along with any other files, data, information, and/or tables for holding, for example, hardware identifiers. This other executable is the final digital information product.

The first part of the digital information product, i.e., the executable stub, would load and execute normally like any other application. It then would perform any authorization procedures. Once the proper authorization or password was completed successfully, an unwrap procedure would be performed as will now be described in connection with Fig. 8, it would then allocate enough protected memory using a function like VirtualAlloc() as shown in step 150:

```
DWORD nFileSize = 0;
DWORD nPhantomFileSize = 0;
```

```
DWORD exeOffset = 0;
DWORD nPreferredLoadAddress = GetPreCompiledLoadAddress();
CString cCommandFile = UnwrapGetNTCommandFile();
exeOffset = UnwrapGetDllOffset(cCommandFile);
nFileSize = UnwrapGetDllSize(cCommandFile);
nPhantomFileSize = nFileSize + 0x3000; // add any needed extra space
// Increase buffer size to account for page size (currently Intel page size).
DWORD nPageSize = GetPageSize();
nPhantomFileSize += (nPageSize -(nPhantomFileSize % nPageSize));
// Allocate the memory to hold the decrypted executable.
LPVOID lpvBlock = VirtualAlloc((LPVOID) nPreferredLoadAddress,
                nPhantomFileSize,
                        MEM_RESERVE | MEM_COMMIT, PAGE_READWRITE);
```

This function can request a particular address space. Preferably, this address space is the preferred load address space to which the .dll was linked in order to minimize any needed relocation and fix up code. The stub executable may lock that area of memory in step 152, for example by using VirtualLock() to prevent any memory writes to a swap file, depending on the operating system, as shown below:

```
BOOL bVLock = VirtualLock((LPVOID) nPreferredLoadAddress, nPhantomFileSize);
```

The memory area still should be secure even without this preventive step since the Windows 95 and NT operating systems do not allow any user access to swap files.

The encrypted code is then copied from the digital information product into the allocated protected memory in step 154, for example by using the following command:

```
UnwrapCopyHiddenExeToMem(cCommandFile, exeOffset, nFileSize, (char *) lpvBlock);
```

Once in memory, the stub would then decrypt the code to that same portion of memory in step 156, for example by using the following commands:

```
CwrapDecryptSeed(cPassword.GetBuffer(0), cPassword.GetLength());
CwrapDecrypt((unsigned char *) lpvBlock, 0, nFileSize);
```

Any "fix up and relocation" type services would then be performed in step 158, for example by using the following command:

```
UnwrapFixUpAndRelocateDll(lpvBlock);
```

Possibly, the memory protection may be changed to execute only in step 160, for example by using the VirtualProtect() command as follows:

```
DWORD lpflOldProtect; // variable to get old protection
BOOL bVProtect = VirtualProtect((LPVOID) nPreferredLoadAddress,
                                nPhantomFileSize,
                                PAGE_EXECUTE,
                                &lpflOldProtect);
```

Function calls then can be made into that area of memory that now contains the decrypted code:

```
UnwrapDoDllAlgorithms();
```

Some of the "fix up" operations to be performed above include placing the addresses of external or stub.exe functions into the address place holders of the decrypted .dll or internal code, by using commands similar to the following:

```
WriteAddress((char*) 0x0a406104, (DWORD) &CallBackFunction1);
WriteAddress((char*) 0x0a406100, (DWORD) &CallBackFunction2);
```

For instance a wrapper function could be created in the outer stub.exe that received a size parameter, allocated that amount of memory off of the heap, and passed back the starting address of that block of memory. Another example would be to have encrypted algorithms within the hidden, encrypted .dll which would be called at run time from the front end stub once decrypted

within protected memory. The dynamic link library would be compiled and linked to expect a pointer to a function that took that parameter and/or returned a value by including prototypes in the header file as follows:

```
void (*lpCallBackFunc1)();
void (*lpCallBackFunc2)(unsigned long);
```

Function calls to "external" functions also could be added as follows:

```
(*lpCallBackFunc1)();
unsigned long z = x * x;
(*lpCallBackFunc2)(z);
```

At run time the "fix up" code would take the run time address of that "wrapper function" and place it into the pointer address within the .dll block of code as follows:

```
WriteAddress((char*) 0x0a406104, (DWORD) &CallBackFunction1);
WriteAddress((char*) 0x0a406100, (DWORD) &CallBackFunction2);
```

This information is readily available using the .cod output files from the compiler, an example of which follows:

```
_TestSum PROC NEAR                                          ; COMDAT
; Line 8
 00000    56          push   esi
; Line 23
 00001    ff 15 00 00 00
          00          call   DWORD PTR_lpCallBackFunc1
; Line 24
 00007    8b 44 24 08 mov    eax, DWORD PTR_a$[esp]
 0000b    50          push   eax
 0000c e8 00 00 00 00 call   _TestSquare
```

```
00011      83 c4 04       add    esp, 4
00014      8b f0          mov    esi, eax
; Line 25
00016      8b 44 24 0c    mov    eax, DWORD PTR _b$[esp]
0001a      50             push   eax
0001b      e8 00 00 00 00 call   _TestSquare
00020      83 c4 04       add    esp, 4
00023      03 c6          add    eax, esi
; Line 28
00025      5e             pop    esi
00026      c3             ret    0
_TestSum ENDP
_TEXT      ENDS
;      COMDAT _TestSquare
_TEXT      SEGMENT
_x$ = 8
_TestSquare PROC NEAR                                    ; COMDAT
; Line 30
00000      56             push   esi
; Line 32
00001      8b 74 24 08    mov    esi, DWORD PTR _x$[esp]
00005      0f af f6       imul   esi, esi
; Line 34
00008      56             push   esi
00009      ff 15 00 00 00
           00             call   DWORD PTR _lpCallBackFunc2
0000f 83 c4 04            add    esp, 4
00012      8b c6          mov    eax, esi
; Line 36
00014      5e             pop    esi
00015      c3             ret    0
_TestSquare ENDP
```

Such information also is available from .map output files from the linker where the "f" between the address (i.e., 0a406100) and the object file (i.e. Algorithms.obj) means it is a "flat" address (i.e., hard coded by the linker) and the lack of an "f" means that it is an address pointer to be supplied at run time (load time) where the address that is contained in that address location is used and not the actual address location (i.e., the address that is contained at address location 0a406100 and not 0a406100 itself):

```
0001:00000000      _TestSum                0a401000 f Algorithms.obj
0001:00000030      _TestSquare             0a401030 f Algorithms.obj

0003:00001100      _lpCallBackFunc2        0a406100   Algorithms.obj
0003:00001104      _lpCallBackFunc1        0a406104   Algorithms.obj
```

When the code inside the .dll makes a "call" to a dereferenced pointer, it would jump to the correct function in the outer code and return the expected return value (if any). For example:

```
void CallBackFunction1(){
// This is the first function that exists in the Stub executable
// whose address has been placed at the appropriate location inside the "dll" code
// that has now been decrypted in a block of memory. The code inside the "dll"
// makes a function call to this function. In its encrypted state, the "dll" does not contain
// this address, but merely has a placeholder for the address. The “dll” has enough space
allocated to hold an
// address of this size. After the “dll” has been decrypted at run time, its address is
// placed in that location so the code inside the "dll" that references (or more
// appropriately dereferences) that address can jump (which is function call) to this
// address.
AfxMessageBox(
_T("This is the FIRST Stub.exe call back function being called from the dll."));
        return;
}
```

```
void CallBackFunction2(DWORD nNumber){
// See comment for CallBackFunction1 except this function receives a parameter off
// of the stack.  It could also return a value as well.
        CString
        cString(
T("This is the SECOND Stub.exe call back function being called from the dll"));

        har buffer[20];
        ltoa(nNumber, buffer, 10);

        cString += _T(" with a parameter of ");
        cString += buffer;
        cString += _T(".");
        AfxMessageBox(cString.GetBuffer(0));
        return;
}
```

The outer stub.exe would make the same kinds of jumps or function calls into the now protected decrypted code block as follows:

```
DWORD c;

// This command declares a function pointer.  This command is different for different function
// calls.  Here the called function takes two integer parameters and
// passes back a DWORD.
DWORD (*lpFunc)(DWORD,DWORD);

// The function pointer is then pointed to the starting address of the function in the
// block of memory that now holds the decrypted DLL.
lpFunc = (DWORD (*)(DWORD,DWORD)) UnwrapFixUpAndRelocateDll();

// Now call that "function" which is really like all function calls, i.e., a jump to
```

```
// the address where that function exists. In this case, two
// variables are passed to that function and returning a value from that function. This function
illustrates that the function call
// can be more complicated than merely a simple jump
// to an address. Inline assembler code may be used to push the variables onto
// the stack frame and return the variable from the eax register, but this function enables
// the C++ compiler to do the same function.
c = (DWORD) (*lpFunc)(a, b);
```

This mechanism requires the unwrap procedure and the now decrypted code to have intimate knowledge about procedural interfaces of each other but no knowledge about each other's implementation. This is the way most executable .exe files and .dll files behave but with the addition of a series of "wrapper" functions on either side for communication. This method works under Windows 95 and Windows NT 4.0 operating systems and should work under Windows NT 3.51 and other operating systems.

Another modified version of this mechanism that works under the Windows NT 4.0 operating system because of functions specific to Windows NT 4.0 would be to have another hidden and/or encrypted executable within the digital information product. This executable would be copied to a physical disk in an unencrypted form, launched or loaded with the CreateProcess() command in its current form but called with a parameter to load the executable in suspended mode:

```
BOOL success = CreateProcess(cFrontEndExe.GetBuffer(0), 0, 0, 0, TRUE,
                    CREATE_NEW_CONSOLE | CREATE_SUSPENDED,
                0, 0, &startUpInfo, &processInfo);
```

Then the first process would copy the encrypted dll into its own process and decrypt it, allocate enough memory using VirtualAllocEx() in its current form in the second process that has just loaded the expendable front end executable in a suspended state as follows:

```
LPVOID lpvBlockEx = VirtualAllocEx(processInfo.hProcess,
```

```
(LPVOID) nPreferredLoadAddress, nPhantomFileSize,
MEM_RESERVE | MEM_COMMIT,
PAGE_READWRITE);
```

The decrypted code is copied from the first process to the second suspended process using WriteProcessMemory() in its current form:

```
BOOL bWriteProcessMemory = WriteProcessMemory((HANDLE) processInfo.hProcess,
(LPVOID) lpvBlockEx, (LPVOID) nPreferredAddress,
(DWORD) nPhantomFileSize, (LPDWORD) &nBytesWritten);
```

The primary thread of the previously launched second process is then resumed:

```
DWORD nResumed = ResumeThread(processInfo.hThread);
```

Any necessary function pointers are then placed in the correct locations by the second process, the area of memory is locked to prevent any writes to a swap file, and the memory protection is changed to execute only as follows:

```
WriteAddress((char*) 0x0a406104, (DWORD) &CallBackFunction1);
WriteAddress((char*) 0x0a406100, (DWORD) &CallBackFunction2);

BOOL bVLock = VirtualLock((LPVOID) nPreferredLoadAddress, nPhantomFileSize);
DWORD lpflOldProtect; // variable to get old protection
BOOL bVProtect = VirtualProtect((LPVOID) nPreferredLoadAddress,
nPhantomFileSize, PAGE_EXECUTE, &lpflOldProtect);
```

The program can continue running by making and receiving calls to and from the decrypted dynamic link library that now resides in the protected memory of its process using commands such as the following:

```
DWORD c;
```

```
DWORD (*lpFunc)(DWORD,DWORD);
lpFunc = (DWORD (*)(DWORD,DWORD)) ExpendableGetEntryAddress();
c = (DWORD) (*lpFunc)(a, b);
```

The first process can either close down or launch another instance of that same process.

In either of these implementations using the same process or launching into a second process, the hidden encrypted code never passes through a file system driver or memory resident program in decrypted form. Code can be split up among different dynamic link libraries so that no two would reside in memory at the same time in order to protect code further. Both of these systems can be implemented using the Win32 function calls. If additional functions, similar to a CreateProcess() command or a LoadLibrary() command but that take a process identifier and address location in memory to load in an executable instead of a physical file, are provided in an operating system then the entire executable and dynamic link library can be hidden, encrypted, and protected on the physical disk and then decrypted within protected memory and use the operating system loader to load it directly to the operating system from memory without residing in decrypted form on any physical medium.

Having described the operation and use of the computer program product in accordance with the invention, embodiments of which are described above in connection with Figs. 3-8, and the operation of the unwrap procedure and device driver it contains, the process of constructing such a computer program product will now be described in more detail. Referring now to Fig. 9, an embodiment of this process for creating a computer program product is shown. This process can be applied to any digital information including an arbitrary executable computer program, dynamic link libraries and related files of data. All digital information is treated as mere data by this process. Each separate data file is combined into a single file by this process, with an executable program for performing the unwrap procedure, and optionally executable program code for a virtual device driver, into the computer program product. Each file of hidden information has a unique location and is identified by its own begin and end markers as shown in Fig. 3. The first step of this process is opening a new data file for the computer program using a name that will be used to indicate an executable file (step 200). For example, an executable word processing program may be named "word_processor.exe" in the Windows95 operating system.

The three portions of the computer program product are then inserted into the open data file. First, the unwrap procedure is inserted at the beginning of the file in an executable format in step 202. The begin tag for the optional device driver is then inserted in step 204. The executable device driver program code is then inserted in step 206, followed by its corresponding end tag in step 208. For each hidden file to be inserted into this computer program product, steps 210 to 216 are performed. First, the begin tag is inserted in step 210. The begin tag also may include an indication of a unique name of the file which will be used as its name in the phantom directory created by the unwrap procedure. The hidden file is then encrypted and/or compressed in step 212 and inserted into the data file in step 214. The end tag for the hidden file is then inserted in step 216. The device driver and all of the tags may be encrypted also if the unwrap procedure has suitable decryption procedures. The computer program file is closed when the last hidden file is processed.

Using the present invention digital information, such as executable program code or various kinds of data, is loaded and unloaded as needed, and thus does not take up any more memory than is necessary. At no time does unencrypted digital information, such as computer program code, exist on disk in accessible and complete decrypted form. Because the original digital information is available as a read only file in one embodiment of the invention accessible only to the device driver, the digital information may be accessed over networks, from a CD-ROM or from a DVD, and can be made to have a limited number of uses. This mechanism is particularly useful for controlling distribution of computer programs, digitized movies or other information while reducing the cost of such distribution and control. For example, software may be distributed over a network on a single use basis, and charges may be levied on a per use basis. The ability to reverse engineer an application program also may be reduced.

One benefit with this system over some other systems for preventing unauthorized access to digital information is that the content provider maintains control of the encryption applied to the information how it may be decrypted. Any need for either a centralized facility or a predetermined decryption program is eliminated. An operating systems manufacturer or other platform vendor merely provides the capability for the information to be accessed and decrypted on the fly. Since the valuable information and any other tables of authorization codes, passwords, or hardware identifiers that the content provider may use to secure the information resides in one large encrypted file, it becomes difficult, if not impossible, for someone to determine just where any of this information exists.

A potential scenario with authorization procedure in which the present invention may be used is the following. A consumer purchases a DVD disk containing a movie. The user puts the disk into the player. This is the first time the disk is installed. The content provider's functions are loaded into the DVD chip, which looks in the encrypted table and sees that this is the first time this disk is being played. The player then displays on a screen a numeric identifier and toll free phone number. The consumer calls the toll free phone number and inputs the numeric identifier that was displayed on the screen. The content provider provides a numeric password based on the numeric identifier that the user inputs into the DVD. The content provider may develop a database of information about its consumers that also may be used to detect pirating of the digital information product. Now that this authorization has taken place, the software that the content provider wrote, and is now in the DVD chip, takes a hardware identifier from the DVD and encrypts it and puts it in the encrypted and buried table on the disk. Alteratively, the data may be decrypted in memory and re-encrypted back onto the disk using the hardware identifier as part of a key. Now that disk will run and show the movie and will only run on that DVD and no other. The content provider could allow for a table of hardware id's so they could limit the number of DVD's that disk would run on or a limited number of times it can be shown. It should be understood that many other authorization procedures may be used.

In the foregoing scenario, the movie is encrypted on the same disk inside of the encrypted file that contains the table and functions the content provider distributed. The movie is decrypted by the decryption functions contained in the file directly to the DVD chip. At no time does the movie reside anywhere in decrypted form. The content provider can protect the movie with any desired level of security (for both encryption and authorization).

In the present invention, the onus of protection of content does not reside with a hardware manufacturer or platform provider but in the hands of the content provider. The hardware manufacturer only provides the mechanism to protect the digital information through the operating system. The technique and implementation of protection resides in the hands of the content provider. This mechanism allows the content providers to change the level of security as needed without any modifications to the hardware. The security of the content is provided by the encryption/decryption algorithms, public/private keys, and authorization methods which are determined by the content provider. Even each individual product can have its own encryption/decryption algorithms and/or public/private keys. All of these can be changed and enhanced as the market demands.

The present invention also could be used for on-line or live use of digital information. For example, a movie could be retrieved on demand and recorded by a consumer. A set top box could receive the digital information, decrypt it, and then re-encrypt and store the information using, for example, a hardware identifier of the set top box. Since home movies digitally recorded would be encrypted using the hardware identifier of the device used in recording, that home movie could not be played on another or only on a limited number of other devices and/or for only a specified number of times depending on the wishes of the content provider. Since the algorithms are downloaded at the time of recording from a service provider, e.g., the cable company, the content provider (movie company) would provide the encrypted data to the service provider to present to their customers. The service provider need not be concerned with the encryption/decryption and authorization functions used by the content provider. Similar uses are possible with other data transmission systems including, but not limited to, telephone, cellular communications, audio transmission including communication and the like.

In another embodiment, the stub executable program is a first process that is implemented similar to a debugging tool such as the SoftIce debugger from NuMega Technologies or the WinDebug debugger from Microsoft Corporation for Ring 0 kernel level debugging for an Intel processor based architecture, or the CodeView debugger for ring 3 application level debugging. Such a debugger controls execution of a program to be debugged as a second process and steps through each program statement or opcode of the debugged program. The debugging tool could be modified to monitor each opcode that indicates a jump to a program fragment, such as each instruction or a block code. If the program fragment to be executed is not decrypted, the modified debugger decrypts the program fragment before the jump command is allowed to execute. Each program fragment may be re-encrypted after execution. Clearly, unnecessary debugging commands may be omitted from the modified debugger.

Having now described a few embodiments of the invention, it should be apparent to those skilled in the art that the foregoing is merely illustrative and not limiting, having been presented by way of example only. Numerous modifications and other embodiments are within the scope of one of ordinary skill in the art and are contemplated as falling within the scope of the invention as defined by the appended claims and equivalent thereto.

CLAIMS

1. A computer-implemented process for executing encrypted computer program logic while maintaining protection against copying of corresponding decrypted executable computer program logic, wherein the encrypted computer program logic is stored in association with first executable computer program logic, the process comprising the steps of:

through an operating system of a computer, reading, loading and executing the first executable computer program logic as a first process having a protected memory area defined by the operating system;

the first process decrypting the encrypted computer program logic into second executable computer program logic and storing the second executable computer program logic in the protected memory area; and

the first process causing loading and execution of the decrypted second computer program logic in the protected memory area.

2. The process of claim 1, wherein the encrypted computer program logic and the first executable computer program logic are stored in a single data file accessible through the operating system.

3. The process of claim 1, wherein the execution of the decrypted second computer program logic is performed as a second process having a second protected memory area defined by the operating system.

4. A digital information product including a computer readable medium having digital information stored thereon, the digital information including computer program logic defining first executable computer program logic, wherein the first executable computer program logic when executed performs the following steps:

storing the encrypted computer program logic in a data file accessible through an operating system of a computer, wherein the data file also includes first executable computer program logic;

through the operating system, reading, loading and executing the first executable computer program logic from the data file as a first process having a protected memory area;

the first process decrypting the encrypted computer program logic into second executable computer program logic and storing the second executable computer program logic in the protected memory area; and

the first process causing loading and execution of the decrypted second computer program logic in the protected memory area.

5. A computer system comprising:

a processor for executing computer program logic;

a main memory operatively connected to the processor for storing digital information including executable computer program logic at memory locations addressed by the processor; and

an operating system defined by executable computer program logic stored in the memory and executed by the processor and having a command which when executed by the processor defines means for creating a process in response to a request specifying a process identifier and a memory location in the main memory, wherein the process identifier indicates the process making the request and the memory location stores executable computer program logic which when executed defines the process.

6. A computer system having an operating system, for decrypting digital information, comprising:

means for storing the encrypted computer program logic in a data file accessible through the operating system, wherein the data file also includes first executable computer program logic;

means, invokable through the operating system, for reading, loading and executing the first executable computer program logic from the data file as a first process having a protected memory area;

the first process defining means for decrypting the encrypted computer program logic into second executable computer program logic and storing the second executable computer program logic in the protected memory area; and

the first process defining means for causing loading and execution of the decrypted second computer program logic in the protected memory area.

7. The computer system of claim 6, wherein the encrypted computer program logic and the first executable computer program logic are stored in a single data file accessible through the operating system.

8. The computer system of claim 6, wherein the execution of the decrypted second computer program logic is performed as a second process having a second protected memory area defined by the operating system.

9. A digital information product, including a computer readable medium with computer readable information stored thereon, wherein the computer readable information comprises:

a first portion of executable computer program logic; and

a second portion of encrypted digital information; and

wherein the first portion of executable program logic, when executed, defines means, operative in response to requests for digital information, for accessing the second portion of encrypted digital information, for decrypting the encrypted digital information, and for outputting the decrypted digital information.

10. The digital information product of claim 9, wherein the encrypted digital information is encrypted executable computer program logic.

11. A computer program product including a self-decrypting encrypted executable computer program, comprising:

a computer readable medium having computer program logic stored thereon, wherein the computer program logic defines:

a first module,

a second module,

wherein the first module, when executed by a computer, defines means for loading the second module into memory of the computer, and

a third module defining the encrypted executable computer program,

wherein the second module, when executed by a computer, defines means for communicating with an operating system of the computer to receive requests for program code from the encrypted executable computer program from the third module, and for processing the

requests to access and decrypt the encrypted executable computer program and for providing the decrypted executable code from the third module to the operating system.

12. A process for executing encrypted executable computer programs on a computer system having a processor, memory and operating system, comprising the steps of:

receiving computer program logic having a first module defining a start up routine, a second module, and a third module containing the encrypted executable computer program;

executing the first module of the received computer program logic using the processor,

wherein the step of executing causes the second module to be loaded into the memory of the computer system, and

generating requests from the operating system for data from the encrypted executable computer program which are received by the second module, and

accessing and decrypting the encrypted executable computer program and returning the decrypted executable computer program to the operating system.

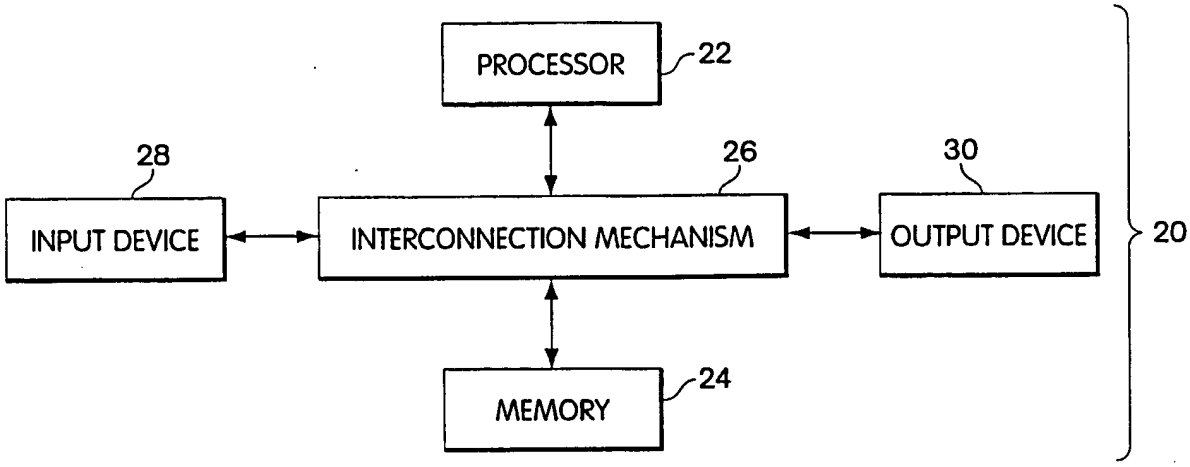

PROCESSOR
22
28
26
30
INPUT DEVICE
INTERCONNECTION MECHANISM
OUTPUT DEVICE
20
MEMORY
24

Fig. 1

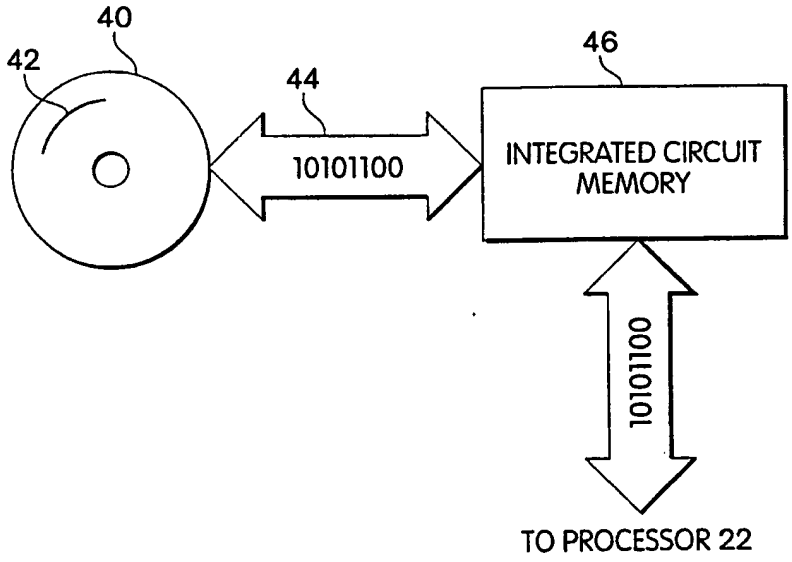

40
42
44
46
10101100
INTEGRATED CIRCUIT MEMORY
10101100
TO PROCESSOR 22

Fig. 2

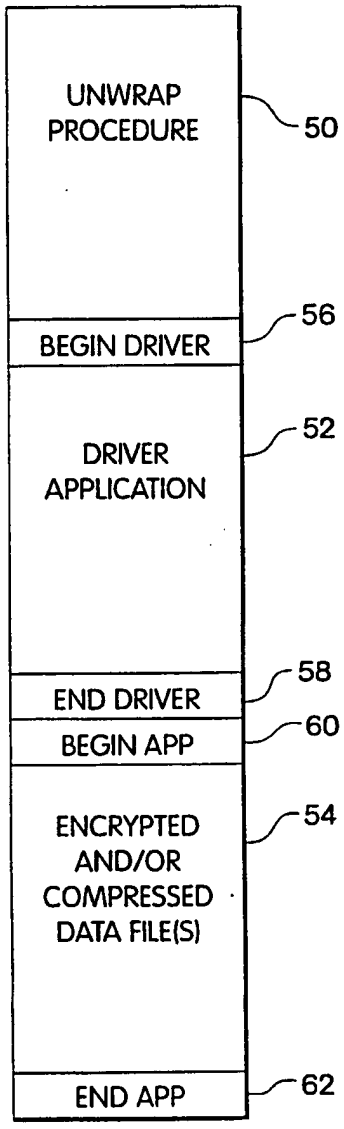

UNWRAP PROCEDURE
50
BEGIN DRIVER
56
DRIVER APPLICATION
52
END DRIVER
58
BEGIN APP
60
ENCRYPTED AND/OR COMPRESSED DATA FILE(S)
54
END APP
62

Fig. 3

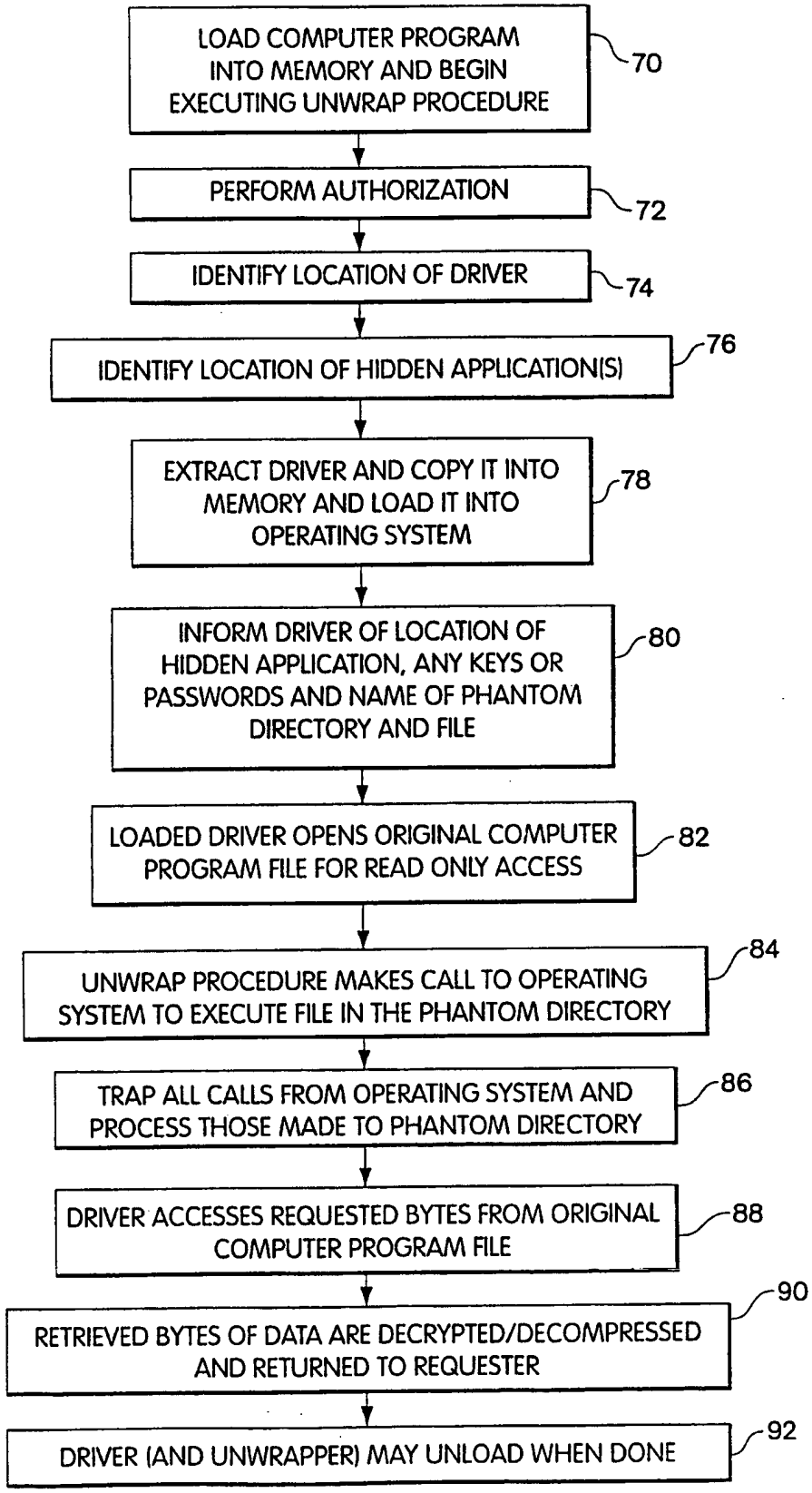

LOAD COMPUTER PROGRAM INTO MEMORY AND BEGIN EXECUTING UNWRAP PROCEDURE
70
PERFORM AUTHORIZATION
72
IDENTIFY LOCATION OF DRIVER
74
IDENTIFY LOCATION OF HIDDEN APPLICATION(S)
76
EXTRACT DRIVER AND COPY IT INTO MEMORY AND LOAD IT INTO OPERATING SYSTEM
78
INFORM DRIVER OF LOCATION OF HIDDEN APPLICATION, ANY KEYS OR PASSWORDS AND NAME OF PHANTOM DIRECTORY AND FILE
80
LOADED DRIVER OPENS ORIGINAL COMPUTER PROGRAM FILE FOR READ ONLY ACCESS
82
UNWRAP PROCEDURE MAKES CALL TO OPERATING SYSTEM TO EXECUTE FILE IN THE PHANTOM DIRECTORY
84
TRAP ALL CALLS FROM OPERATING SYSTEM AND PROCESS THOSE MADE TO PHANTOM DIRECTORY
86
DRIVER ACCESSES REQUESTED BYTES FROM ORIGINAL COMPUTER PROGRAM FILE
88
RETRIEVED BYTES OF DATA ARE DECRYPTED/DECOMPRESSED AND RETURNED TO REQUESTER
90
DRIVER (AND UNWRAPPER) MAY UNLOAD WHEN DONE
92

Fig.4

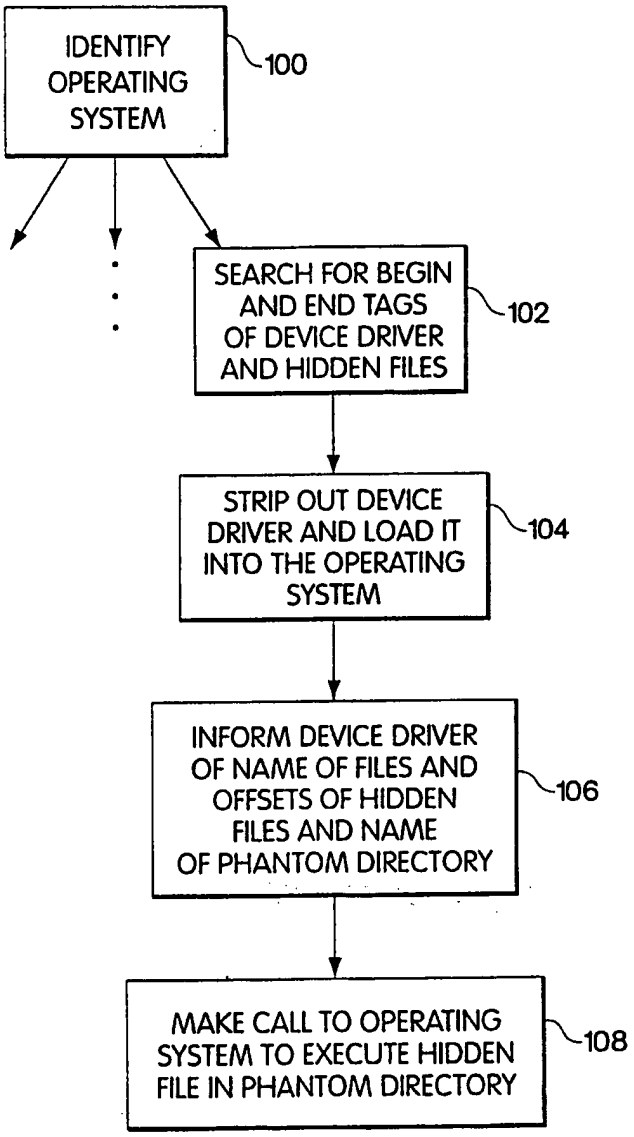

IDENTIFY OPERATING SYSTEM
100
SEARCH FOR BEGIN AND END TAGS OF DEVICE DRIVER AND HIDDEN FILES
102
STRIP OUT DEVICE DRIVER AND LOAD IT INTO THE OPERATING SYSTEM
104
INFORM DEVICE DRIVER OF NAME OF FILES AND OFFSETS OF HIDDEN FILES AND NAME OF PHANTOM DIRECTORY
106
MAKE CALL TO OPERATING SYSTEM TO EXECUTE HIDDEN FILE IN PHANTOM DIRECTORY
108

Fig. 5

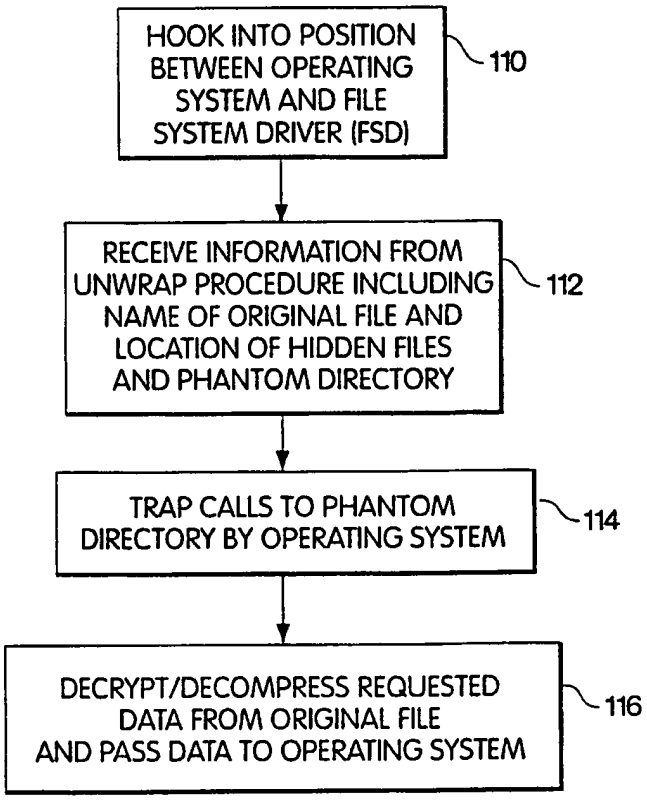

HOOK INTO POSITION BETWEEN OPERATING SYSTEM AND FILE SYSTEM DRIVER (FSD)
110
RECEIVE INFORMATION FROM UNWRAP PROCEDURE INCLUDING NAME OF ORIGINAL FILE AND LOCATION OF HIDDEN FILES AND PHANTOM DIRECTORY
112
TRAP CALLS TO PHANTOM DIRECTORY BY OPERATING SYSTEM
114
DECRYPT/DECOMPRESS REQUESTED DATA FROM ORIGINAL FILE AND PASS DATA TO OPERATING SYSTEM
116

Fig. 6

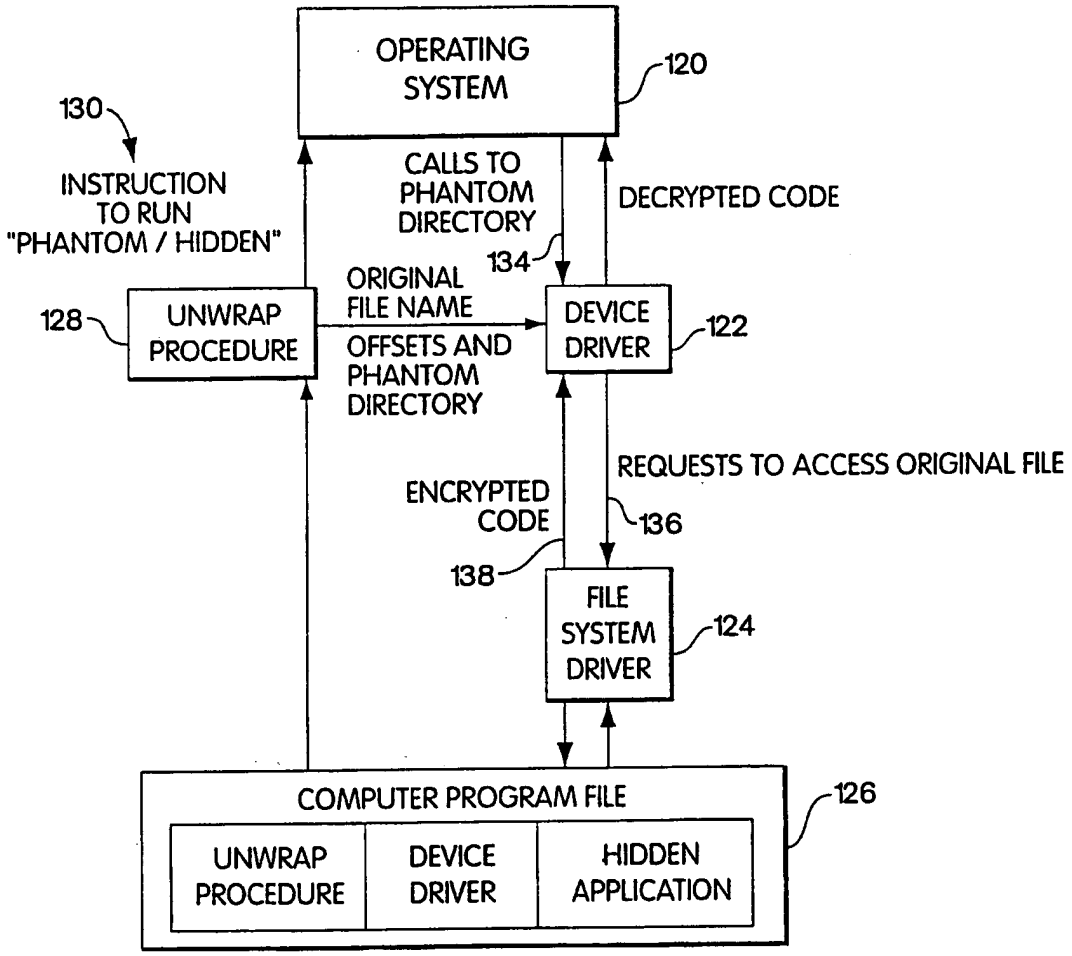

OPERATING SYSTEM
120
130
INSTRUCTION TO RUN "PHANTOM / HIDDEN"
CALLS TO PHANTOM DIRECTORY
134
DECRYPTED CODE
ORIGINAL FILE NAME
OFFSETS AND PHANTOM DIRECTORY
128
UNWRAP PROCEDURE
DEVICE DRIVER
122
REQUESTS TO ACCESS ORIGINAL FILE
ENCRYPTED CODE
136
138
FILE SYSTEM DRIVER
124
COMPUTER PROGRAM FILE
126
UNWRAP PROCEDURE
DEVICE DRIVER
HIDDEN APPLICATION

Fig. 7

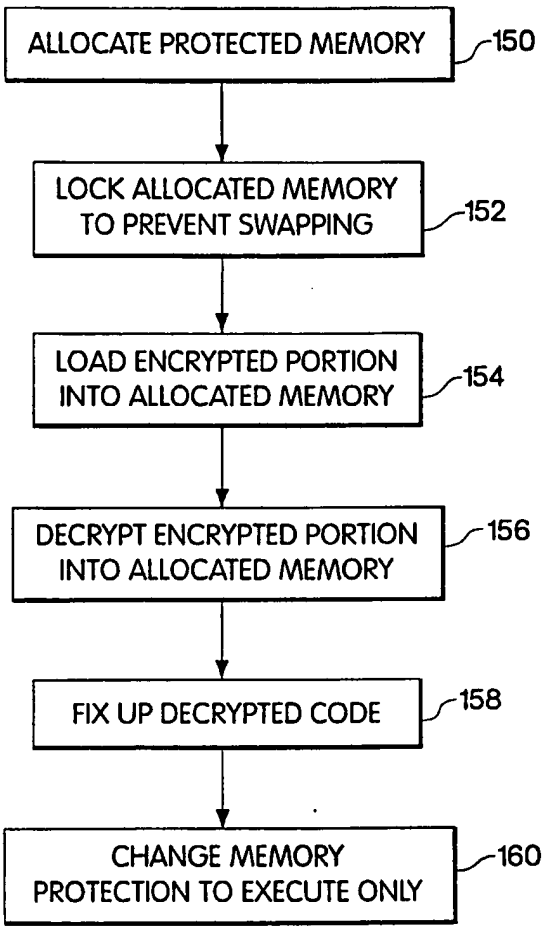

ALLOCATE PROTECTED MEMORY
150
LOCK ALLOCATED MEMORY TO PREVENT SWAPPING
152
LOAD ENCRYPTED PORTION INTO ALLOCATED MEMORY
154
DECRYPT ENCRYPTED PORTION INTO ALLOCATED MEMORY
156
FIX UP DECRYPTED CODE
158
CHANGE MEMORY PROTECTION TO EXECUTE ONLY
160

Fig. 8

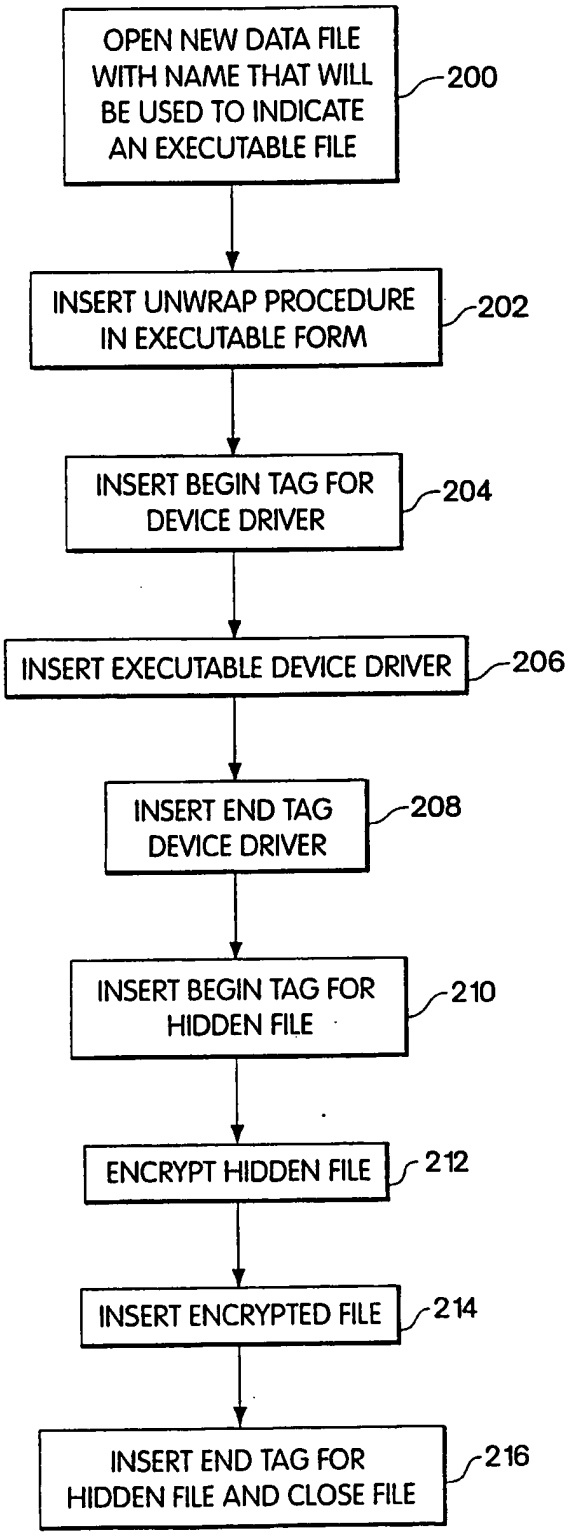

OPEN NEW DATA FILE WITH NAME THAT WILL BE USED TO INDICATE AN EXECUTABLE FILE
200
INSERT UNWRAP PROCEDURE IN EXECUTABLE FORM
202
INSERT BEGIN TAG FOR DEVICE DRIVER
204
INSERT EXECUTABLE DEVICE DRIVER
206
INSERT END TAG DEVICE DRIVER
208
INSERT BEGIN TAG FOR HIDDEN FILE
210
ENCRYPT HIDDEN FILE
212
INSERT ENCRYPTED FILE
214
INSERT END TAG FOR HIDDEN FILE AND CLOSE FILE
216

Fig. 9

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US97/16223

A. CLASSIFICATION OF SUBJECT MATTER
IPC(6) :H04L 9/00
US CL : 380/4
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 380/4,9,23,25,49,50,59

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 4,937,861 A (CUMMINS) 26 June 1990, see Abstract.	1-12
A	US 5,007,082 A (CUMMINS) 09 April 1991, see Abstract.	1-12
A	US 5,144,659 A (JONES) 01 September 1992, see Abstract.	1-12
A	US 5,155,827 A (GHERING) 13 October 1992, see Abstract.	1-12
A	US 5,396,609 A (SCHMIDT et al) 07 March 1995, see Abstract.	1-12

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search
20 JANUARY 1998

Date of mailing of the international search report
18 FEB 1998

Name and mailing address of the ISA/US
Commissioner of Patents and Trademarks
Box PCT
Washington, D.C. 20231
Facsimile No. (703) 305-3230

Authorized officer
BERNARR EARL GREGORY
Telephone No. (703) 306-4153

Form PCT/ISA/210 (second sheet)(July 1992)★